Universidad Autónoma de Madrid
Escuela politécnica superior

Máster Universitario en Investigación e Innovación en Tecnologías de la Información y las Comunicaciones (i^2-TIC)

Trabajo Fin de Máster

ESTUDIO Y ANÁLISIS DE DIFERENTES TÉCNICAS DE MODULACIÓN EN LA ADQUISICIÓN DE UN ODORANTE EN EL CONTEXTO DE NARICES ELECTRÓNICAS

Autor: Ángel Fuente Ortega
Tutor: Francisco de Borja Rodríguez Ortiz

Septiembre 2018

ESTUDIO Y ANÁLISIS DE DIFERENTES TÉCNICAS DE MODULACIÓN EN LA ADQUISICIÓN DE UN ODORANTE EN EL CONTEXTO DE NARICES ELECTRÓNICAS

Autor: Ángel Fuente Ortega
Tutor: Francisco de Borja Rodríguez Ortiz

Grupo de Neurocomputación Biologica (GNB)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Septiembre 2018

Resumen

Resumen

En este Trabajo de Fin de Máster se propone el análisis de una base de datos compuesta por diferentes odorantes, la cual se constituirá utilizando una plataforma de experimentación propia durante el desarrollo de este TFM, mediante el uso de diferentes técnicas de captura basadas en la modulación de la temperatura. Asimismo se han ampliado las funciones del software de captación, junto con la implementación y aplicación de diferentes técnicas de tratamiento, empleadas en los datos durante su clasificación y la codificación de una interfaz gráfica, que permite la interacción con la plataforma de forma visual. Mediante el análisis de la base de datos se demuestran las ventajas de las técnicas de modulación de la temperatura del sensor, frente al uso de una temperatura estacionaria durante la captura de odorantes.

Para realizar este proyecto ha sido necesario la creación de una base de datos, a partir de los datos obtenidos en los experimentos realizados. Su creación se ha realizado bajo diferentes condiciones, variando los algoritmos y los parámetros empleados. La creación de la base de datos ha supuesto un periodo de 3 meses, utilizando dos algoritmos de experimentación distintos: el algoritmo de experimentación con temperatura constante y el algoritmo de adquisición mediante temperatura variable y codificación en amplitud.

El software de la plataforma ha sufrido varias modificaciones, haciéndolo más versátil y fácil de utilizar. Se ha añadido un nuevo algoritmo de captura basado en el uso de un bucle cerrado PID. También existe la posibilidad de configurar la plataforma antes de realizar los experimentos, variando parámetros de captura que inicialmente eran estáticos. Finalmente, se ha codificado una interfaz gráfica, permitiendo configurar los experimentos de forma visual.

Por último, se han estudiado y codificado diversas técnicas para la calibración, atenuación y eliminación de posibles ruidos o distorsiones presentes en las señales capturadas, que dificultan la identificación de los odorantes. Para la clasificación se han utilizado varias técnicas de extracción de características, codificadas durante el desarrollo de este trabajo, con la intención de obtener la mejor forma de clasificar los datos para futuros experimentos.

Como resultados obtenidos al finalizar este TFM se ha conseguido demostrar, en una primera aproximación, que la modulación de la temperatura en la captura de datos ayuda a conseguir mejores resultados en diversos casos, en comparación con la captura de odorantes sin aplicar ninguna técnica de modulación. También se ha demostrado que su combinación con el paradigma temporal a la hora de clasificar puede mitigar, en parte, el efecto provocado por el drift del sensor.

Palabras Clave

Nariz artificial, software de experimentación, técnicas de modulación de la temperatura, plataforma de experimentación, tratamiento de datos, clasificación de odorantes, calibración muestras, extracción de características, drift sensor, SVM, clasificación de patrones.

Abstract

In this master's end job we propose the analysis of a databae composed of different odorants, which will be constituted using an own experimentation platform during this work, using diferents tecniques of experimentation based on the modulation of the temperature. Likewise the functions of the software of experimetation have also been expanded together with the implementation and application of different treatment techniques, which have been applied to the data during the clasification and the coding of a Graphic User Interface. Through data analysis from the database it is demonstrated the advantages of modulations techniques of sensor's temperature, against the use of a stacionary temperature during the capture of odorants.

In order to carry out this proyect, it was neccesary to create a database base on the data obtained in the experiments carried out. It's capture has been carried out under differents conditions varing the algorithms and parameters used. Creation of the database has meant a period of 3 months, using two diferrents experiments algorithms: experimentation algorithm with constant temperature and the algoritm of acquisition by variable temperature and amplitude conding.

Software of the platform has suffered several modifications, making it more versatile and easy to use. A new experimentation algoritm based on the use of a close PID loop has been added. Also exists the possibility of configure the platform before performing the experiments, changing parameters that were initialy statics. Finally, a graphic user interface has codified, allowing to configure the experiments in a visual way.

By last, several techniques have been studied and codified for the calibration, attenutation and elimination of possible noises or distorsions present in the captured signals, which dificults the identifications of the odorants. Several characterization techniques have been used for the clasification, codified during the development of this job, with the point of obtain the best way to classify data for future experiments.

As a results obtained, after the completion of this work, it has been demonstrated, in a first approximation, that the temperature modulation in the data capture helps to achieve better results in various cases, compared to the capture of odorants applying any modulation technique. It has also been shown that its combination with the temporal paradigm at the time of classifying can mitigate, in part, the effect caused by the sensor's drift.

Key words

Artificial nose, experimentation software, temperature modulations techniques, experimentation platform, data processing, classification of odorants, calibration of samples, extraction of characteristics, sensor drift, SVM, pattern classification.

Agradecimientos

Ante todo deseo dar las gracias a mi familia primero, a mis padres y a mis hermanos por estar ahí siempre y ayudarme a acabar este trabajo junto con mis primos y mis tíos, en especial a mi tía Clara.

A mis amigos del colegio y del instituto: Borja, David, Ricardo, Sara, Iván y Cristian también quiero darles las gracias por estar ahí y pedirles perdón por no estar tanto tiempo con ellos como se merecían. Mis amigos de carrera: Patricia, Óscar, Silvia, Alejandro y Mario también se merecen mi agradecimiento por los mismos motivos. No quisiera olvidar a mis amigos del pueblo, a los que conozco desde hace mucho y con los cuales, sobre todo en verano, me lo paso tan bien.

A las personas que he conocido en el grupo GNB y a las que considero mis amigos: Manuel, Roy, Irene, Ángel, Carlos, Aarón, Vinicio, Jessica, Guillermo, María y Marcos, gracias por ayudarme y aguantarme en momentos difíciles, que no han sido pocos.

Por último, agradecer la ayuda y el apoyo de mi tutor Francisco, que me ha guiado por el mundo de las narices artificiales; mundo que me ha gustado más de lo que esperaba.

Índice general

Índice de Figuras

Índice de Tablas

Glosario

- **BBB**: Beagle Bone Black
- **GNB**: Grupo de Neurocomputación Biológica
- **Drift**: Obtener distintos valores en las mismas condiciones debido a la memoria del sensor
- **ADC**: Analog to Digital Converter
- **PWM**: Pulse With Modulation
- **GPIO**: General Purpose Input/Output
- **Experimento**: Medición de los odorantes en una secuencia determinada
- **SO**: Sistema operativo
- **RP**: Recurrence Plot
- **RQA**: Recurrence Quantification Analysis
- **TyH**: Temperatura y Humedad ambientales

1 Introducción

1.1. Motivación del proyecto

El siguiente Trabajo de Fin de Máster se centra en el análisis y estudio de muestras de odorantes obtenidas utilizando la plataforma de captación de odorantes del GNB (Grupo de Neurocomputación Biológica), mediante el uso de diferentes algoritmos de captación de odorantes para su clasificación posterior; así como la mejora y extensión de los códigos de la plataforma, los cuales permiten la captura de datos de forma fácil y rápida, sin necesidad de conocer el hardware que compone la plataforma, ni su funcionamiento interno. Estos cambios han permitido transformar la versión primaria de la plataforma en un instrumento de captura completamente versátil y usable para la captura de odorantes.

El sistema olfativo es capaz de captar una gran cantidad de odorantes. Estos son captados en los receptores que se encuentran en la cavidad nasal. El olor, según la RAE, es la impresión que los efluvios producen en el olfato (Española, 2010). Un olor es una combinación de distintos gases, que según la composición influyen más o menos en la percepción del olfato.

En la actualidad, las narices electrónicas (EN's, Electronic Noses), también llamadas olfatómetros, se están utilizando para la exploración de diversos odorantes en distintos ámbitos, teniendo un amplio nicho de usos que van desde la medición de la calidad del aire, pasando por su incorporación en sistemas médicos, hasta la contabilización de seres vivos en una sala donde no se pueda introducir una cámara, entre otros muchos. Estos olfatómetros se han desarrollado inspirándose en la modulación propia del sistema olfativo de los seres vivos, donde se utilizan la variación de la temperatura y la succión para llevar a cabo la discriminación de los odorantes (Vergara et al., 2005; Yáñez et al., 2012; Vergara et al., 2007; Herrero-Carrón et al., 2015). Actualmente, su investigación supone un reto debido al amplio abanico de tareas y problemas que las rodean. Algunos de los temas que se están investigando en el ámbito de las narices artificiales son: la búsqueda de la mejor técnica de adquisición de odorantes, junto con aquellos parámetros para los cuales se consigue una discriminación óptima de los odorantes analizados (Vergara et al., 2007; Ngo et al., 2007) y la búsqueda de una fuente de odorante junto con la dispersión de la pluma de odorante en el ambiente (García Saura et al., 2014; Vázquez Rubio et al., 2013; Pequeño Zurro, 2015). También se estudia como eliminar las interferencias, tanto exteriores como interiores, que distorsionan la señal que leemos del sensor, obteniendo así una señal más limpia (Marco and Gutierrez-Galvez, 2012; Huerta et al., 2016).

La plataforma utilizada en este proyecto es la desarrollada en el trabajo (Cruz Gutiérrez, 2016) del GNB. Esta plataforma cuenta con varios algoritmos de captura que permiten registrar un mismo odorante de diferentes formas. El coste de fabricación que tiene es bajo, lo que permite poder fabricar otras plataformas que permitan la captura de datos y su análisis posterior para un mayor conjunto de odorantes.

Actualmente en el mercado pueden encontrarse muchos tipos de sensores y dependiendo del modelo utilizado los resultados varían. Los sensores con mayor precio tienen una variabilidad menor en los resultados que generan, pero son menos accesibles, mientras que los sensores con precios más bajos son más accesibles, pero los resultados tienen una variabilidad mayor. Dentro de la amplia gama de modelos y tipos de sensores que existen en el mercado, los sensores utilizados en este trabajo son de tipo MOS (Metal Oxide Semiconductor)(Pearce et al., 2006), que son los más extendidos y comunes. Estos sensores generan una corriente eléctrica debido a las reacciones de oxidación que ocurren en su superficie. Durante su uso, los sensores tienen una temperatura de funcionamiento, también llamada de calentamiento, que en la gran mayoría de los casos es estacionaria, es decir, permanece siempre estable. El uso de una temperatura constante aporta información sobre el odorante que se está captando, pero puede limitar la información que obtenemos de este. En ciertos estudios se ha propuesto la variación de ciertos parámetros para tratar de obtener una cantidad de información mayor a la que se obtiene dejando la temperatura estática (Fonollosa et al., 2013; Nakata et al., 1996; Lee and Reedy, 1999; Martinelli et al., 2013b; Herrero-Carrón et al., 2015).

En los trabajos previos realizados en el GNB (Cruz Gutiérrez, 2016; Ortega, 2017), los experimentos utilizaban la temperatura de calentamiento recomendada por el fabricante (Figaro, 2016b) de un modo constante, actuando así de modo estacionario. Con la variación de la temperatura a la que opera el sensor se alteran las reacciones oxidantes que se producen en la superficie de este, enriqueciendo la respuesta dinámica que obtenemos y mejorando la selectividad y sensibilidad del semiconductor del sensor, con la esperanza de obtener más información para un mismo odorante que utilizando una temperatura estable. Un motivo para la realizar esta variación en la temperatura es que cada gas tiene una relación característica de temperatura/conductancia, por lo que un sensor que varia su respuesta es análogo a un array de sensores con perfiles de temperatura estables a diferentes valores (Fonollosa et al., 2013; Lee and Reedy, 1999).

Para realizar este barrido de la respuesta dinámica se pueden aplicar varias técnicas de variación de la temperatura. La más básica consiste en dejar un perfil de temperatura estable, con la esperanza de que se obtengan los mismos valores para su análisis posterior. También se puede modificar la temperatura del sensor acorde a la sensibilidad que tiene este a un odorante para una determinada temperatura, de tal forma que el sistema se auto-regule, explorando los diferentes estados del sensor (Martinelli et al., 2012, 2013a). Otra técnica consiste en la exploración de los valores máximos y mínimos de la conductancia, utilizando una señal objetivo que va ajustando los valores que recibe del sensor, modificando la señal objetivo y explorando los posibles estados del sensor (Herrero-Carrón et al., 2015). Otro método, para realizar este barrido de temperatura, consiste en explorar los estados del sensor utilizando variaciones súbitas en la amplitud de la señal térmica, generando una respuesta diferente con los cambios de la temperatura (Hossein-Babaei and Amini, 2014, 2012), o mediante la utilización de secuencias aleatorias de diferente longitud, para modular esta temperatura y obtener respuestas, o huellas, características para un mismo odorante (Vergara et al., 2007).

Mediante este barrido de la respuesta dinámica del sensor, se pretende obtener más información para un mismo odorante, que utilizando una temperatura estacionaria. En este contexto, el término modulación de temperatura hace referencia a la variación de la señal de temperatura de adquisición del sensor, a través de los datos que hemos obtenido previamente de este, como puede ser el voltaje.

Además de los diversos parámetros regulables del sensor, que se pueden ajustar hasta conseguir el valor óptimo, también hay valores externos como la humedad y la temperatura, que afectan directamente a las medidas tomadas del sensor (Wang et al., 2010; Buehler and Ryan, 1997). Dependiendo del tipo de sensor que se utilice, estos efectos se pueden paliar, ya sea por el propio diseño del sensor o por el tratamiento de los datos obtenidos (Huerta et al., 2016).

Dependiendo del tipo de sensor utilizado, los experimentos realizados poseen una variación, aún utilizando el mismo sensor y el mismo odorante. Esto se debe principalmente a dos factores: el primero es denominado deriva real y se debe a la interacción del químico del odorante con el propio sensor, produciendo una reacción diferente dependiendo del odorante utilizado, puesto que la composición de los sensores no es exacta entre ellos, y a la contaminación del sensor por parte del odorante. El segundo factor es denominado deriva de segundo orden y se produce por cambios externos e iteraciones con el sistema de captura, como la variación producida por la temperatura y humedad ambiental o la cantidad de volumen de odorante que recibe el sensor a través del sistema de medición entre otros. El conjunto total que produce este efecto se denomina drift, y puede compensarse estandarizando las señales para obtener así una linea base para la clasificación, utilizando técnicas de transferencia de calibración (Vergara et al., 2012).

Para realizar los experimentos de captura, la plataforma cuenta con un software que utiliza los diferentes algoritmos de captura implementados, aunque de un modo muy primario. Con la mejora y ampliación de este se pretende obtener un software versátil y fácil de usar para la captura de odorantes sin tener que entender el hardware de la plataforma, ni tener que cambiar ningún componente de esta. Además de este software, se implementarán diferentes técnicas para el tratamiento de las señales obtenidas, con las que se pretende mitigar el drift del sensor o los efectos de la temperatura y humedad ambientales en las señales obtenidas.

Una vez que se haya realizado el tratamiento de las señales, se procederá a su clasificación, con el fin de obtener la mejor técnica de modulación de entre las que se encuentran implementadas, y los mejores parámetros de captura. Para ello se aplicarán varias técnicas de extracción de características a las señales, con la intención de conseguir la máxima discriminación, además de reducir su espacio de características facilitando la clasificación.

La clasificación de las muestras necesita un base de datos grande, por lo que se realizarán varios experimentos mediante el uso de los diferentes algoritmos de captura para la creación de una base de datos con diferentes odorantes para su posterior clasificación.

Este proyecto parte de la base de otros trabajos anteriores ya realizados en el GNB como: (Cruz Gutiérrez, 2016; Yáñez, 2009; Yáñez et al., 2012; Ortega, 2017). El propósito de este trabajo es la ampliación y mejora de los códigos que componen el software de la plataforma, así como la captura de un conjunto grande de datos mediante el uso de diferentes técnicas de captura de datos para realizar una comparación entre ellas, junto con el uso de varias técnicas de calibración y procesamiento de los datos obtenidos para su posterior clasificación.

1.2. Objetivos

El objetivo de este TFM consiste en el análisis de las muestras de odorantes, obtenidas en los múltiples experimentos realizados. Con esto se pretende conseguir el algoritmo de captura, de entre los múltiples codificados en la plataforma, con el que se obtiene una mejor discriminación de los odorantes, así como los parámetros de experimentación utilizados para los cuales se consiguen los mejores resultados.

Para lograr este objetivo, se modificará el software original de la plataforma, para adaptarlo a las nuevas modulaciones implementadas en esta (Velasco Botina, 2017). Con las futuras modificaciones y extensiones del software se pretende facilitar el uso de la plataforma. Para ello, se

implementarán mejoras en el pseudo-lenguaje, facilitando así la programación de experimentos, además de una interfaz gráfica que permita programar de forma visual todos los experimentos, sin recurrir a la terminal de usuario.

Además de las diferentes técnicas de modulación que dispone la plataforma, como son las modulación en amplitud o en frecuencia (Martinelli et al., 2013a; Hossein-Babaei and Amini, 2012), se han utilizado técnicas avanzadas para el tratamiento posterior de las señales capturadas. Se eliminará el efecto que provoca la temperatura y humedad ambientales en el sensor, ya que al ser sensores de bajo coste estas variaciones les afectan más, utilizando el modelo propuesto en (Huerta et al., 2016). También se estudiarán y codificarán diversas técnicas de extracción de características, que permitan reducir el espacio de características de las señales, para después aplicar técnicas basadas en tecnologías de aprendizaje automático.

Para saber cual de los múltiples algoritmos implementados es el que mejores resultados aporta, es necesario disponer de una gran cantidad de datos, para que los análisis que se realicen con posterioridad sean fiables. Por ello, se dedicará una gran parte del tiempo disponible a la realización de experimentos con los diferentes algoritmos de captura, utilizando diferentes parámetros de captura, para la creación de una base de datos para su uso posterior en los análisis.

1.3. Estructura de la memoria

La memoria de este TFM se encuentra dividida en distintas secciones:

- **Estado del arte:** En esta sección se describe el marco teórico que es necesario entender para realizar el proyecto. La plataforma de captación de odorantes utilizada, el modelo de sensor olfativo junto con el proceso de transducción de un odorante en un pulso eléctrico, la explicación de las diferentes técnicas de modulación utilizadas para la captura de datos, el pseudo-lenguaje de la plataforma y las diferentes técnicas de procesamiento de datos utilizadas para el procesamiento de las señales obtenidas de la plataforma.
- **Sistema y diseño:** Se explica la reestructuración y extensiones de los códigos de experimentación ya desarrollados junto con las variaciones incluidas en el pseudo-lenguaje.
- **Desarrollo:** Se explican todos las extensiones y variaciones implementadas en el software original de la plataforma, así como la implementación de las diferentes soluciones software que se han utilizado en el tratamiento de los datos.
- **Experimentos realizados:** Se explican los experimentos que se han realizado, así como las muestras utilizadas y el proceso de creación de estas.
- **Resultados:** Se explica todas las técnicas aplicadas a los datos, junto con el resultado de las clasificaciones obtenidas mediante el uso de técnicas de machine learning. Además se relata todo el procesamiento realizado a los datos obtenidos mediante la aplicación de diversas técnicas, del contexto de narices electrónicas, implementadas.
- **Conclusiones y trabajo futuro:** En este último capítulo se exponen las conclusiones obtenidas con la realización de este TFM, junto con las posibles vías de continuación para este trabajo.

2

Estado del arte y técnicas de procesamiento de señal implementadas

2.1. Introducción

En este capítulo se enuncian las bases teóricas para entender el trabajo realizado: la plataforma de captación de odorantes utilizada, el software utilizado en la plataforma, el modelo de sensor olfativo junto con todos los algoritmos usados para la captación de odorantes y las diferentes técnicas utilizadas para el tratamiento de las señales obtenidas.

2.2. Plataforma de captación de odorantes

La plataforma de captación de odorantes es la base de este proyecto junto con el software que contiene y permite su uso, ya que el conjunto de ambos permite la captura de odorantes. La plataforma es un proyecto del GNB (Cruz Gutiérrez, 2016) que permite la captación de diversos odorantes de forma simple. Está compuesta de varios circuitos de captación de odorantes mediante diferentes técnicas de adquisición. Los diferentes circuitos de captación que posee la placa son:

- Circuito de adquisición de odorantes con temperatura constante.
- Circuito de adquisición de odorantes con temperatura variable y codificación temporal en amplitud.
- Circuito de adquisición de odorantes con temperatura variable y codificación en frecuencia.

Mediante el uso de estos circuitos, se pueden obtener diferentes muestras de odorantes para su análisis. La plataforma también contiene otros circuitos que sirven de complemento a los datos captados y permiten registrar otros datos o controlar elementos pertenecientes a la plataforma. Esos circuitos son los siguientes:

- Circuito para el control de las electroválvulas.

- Circuito para el control del motor de succión.
- Circuito de registro de temperatura y humedad ambientales.

El circuito de control de las electroválvulas regula que odorantes le llegan al sensor, mediante la apertura de un conjunto de válvulas que permiten el registro de un odorante o una mezcla de varios a través de la apertura de varias válvulas al mismo tiempo. El circuito del motor de succión permite regular el flujo de odorante que le llega al sensor mediante la potencia de funcionamiento que se le otorga al motor. Cuanta mayor potencia se le de al motor, más odorante le llega al sensor registrándose así una señal más intensa, y cuya variación es mayor en un instante de tiempo menor que si el motor funcionase con una potencia menor. El último circuito permite registrar los valores de la temperatura y humedad del ambiente, que pueden usarse para efectuar la decorrelación de la humedad y temperatura de la señal obtenida, eliminando los efectos provocados en los datos captados por el sensor (Huerta et al., 2016).

La placa está compuesta por un sistema embebido BBB (Instrument, 2013), ver figura 2.1. Este sistema registra todas las señales que le llegan de ambos sensores: el que se encarga de la captura de odorantes (TGS 2600), ver sección 2.3, y el que realiza la medición de la temperatura y humedad del ambiente (Adafruit, 2012). Además del registro de las señales que obtiene de ambos sensores, la placa también ejecuta el algoritmo de adquisición de datos; junto con el soporte energético de los diversos circuitos que la componen, exceptuando el circuito de control de las electroválvulas, que tiene alimentación propia.

El recorrido que sigue el odorante es el siguiente: el motor comienza a succionar aquellos odorantes cuyas electroválvulas estén abiertas, creando una mezcla de odorantes si se encuentran varias abiertas, llevando el odorante al sensor para después ser expulsando del circuito, ver figura 2.1. Cuando el odorante llega al sensor, se produce una reacción química produciendo un voltaje, que es captado por la placa controladora mediante un puerto determinado de esta, llamado ADC (Analogic Digital Conversor). Los odorantes se encuentran en unos botes que se van renovando cada cierto tiempo debido al desgaste y que se sitúan en un lugar determinado en la plataforma.

Esta plataforma de experimentación tiene un tamaño reducido, comprobar en la figura 2.1, lo que permite que sea fácilmente portable a otras ubicaciones, además de hacerla altamente versátil, debido al uso de una BBB como sistema de control, ya que se puede programar de forma fácil y versátil varias técnicas de adquisición de datos. Los componentes utilizados para su construcción tienen un coste bajo permitiendo que se puedan construir varias plataformas para la realización de un mayor número de experimentos.

El desarrollo de esta plataforma no supone un gran hito por si mismo, ya que se pueden encontrar otras plataforma de captación de odorantes, como por ejemplo (Macías et al., 2012, 2013). La diferencia radica en que la plataforma del GNB permite modificar la señal térmica del sensor de distintas formas, permitiendo registrar odorantes utilizando diferentes perfiles térmicos. Además de la variación del perfil térmico del sensor de captura de odorantes, también se pueden cambiar los sensores de forma simple, para realizar pruebas con otros modelos o para su sustitución en caso de que se estropeasen, tanto el sensor de captura captura de odorantes como el de registro de la temperatura y humedad ambientales, haciendo a la plataforma aún más versátil.

2.3. Sensor TGS2600

El sensor TGS2600 (Figaro, 2016b) es el modelo usado en la plataforma, permitiendo registrar los diferentes odorantes que se encuentran en el ambiente. Este tipo de sensor es un sensor quimioresistivo de tipo MOS (Metal Oxide Semiconductor) (Pearce et al., 2006). Los sensores

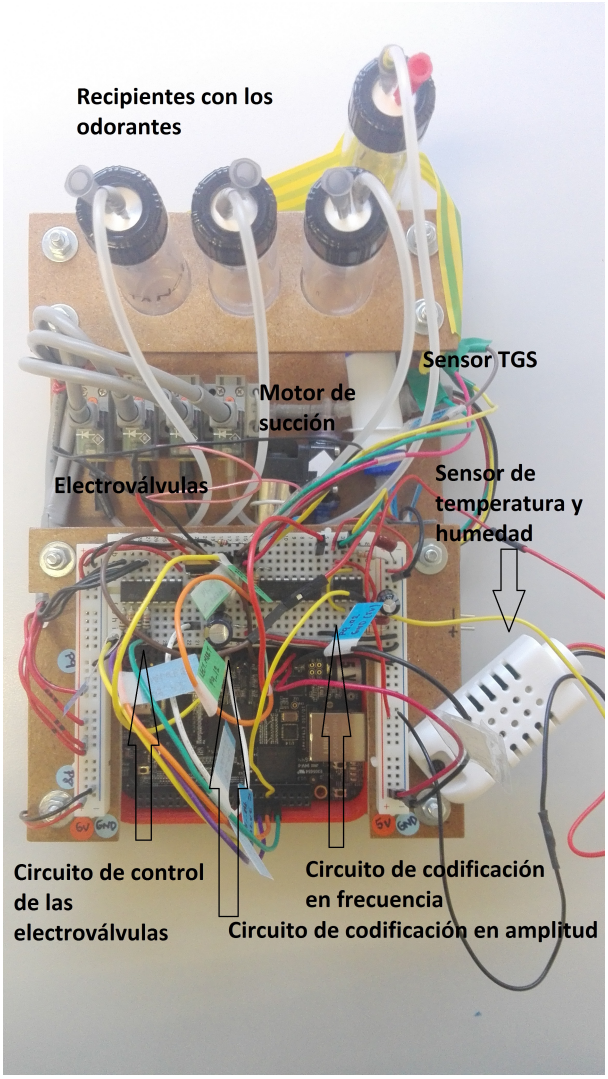
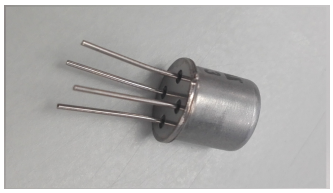


Figura 2.1: Plataforma de captación de odorantes obtenida de (Ortega, 2017).

quimioresistivos varían su impedancia según al compuesto al que se les somete. Están formados por una lámina gruesa de estaño sometida a varios baños químicos con otros compuestos, lo que les permite variar su resistencia en presencia de odorantes. Estos sensores presentan una respuesta lenta ante la presencia de odorantes. En la superficie de la lámina de estaño del sensor se producen reacciones oxidantes, cuyo resultado es la variación en la conductancia/resistencia de este debido a las reacciones de absorción/liberación del odorante. La variación en la rapidez y en la cantidad de reacciones que se producen en el sensor depende de varios factores:

- El odorante al que se expone el sensor.
- La concentración del odorante.
- El material del que está hecho el sensor.
- La temperatura de la superficie del sensor.

En la figura 2.2 puede verse un esquema del sensor. En la imagen de la izquierda puede verse el sensor tal y como es. Consta de cuatro pines de entrada/salida que permiten la comunicación



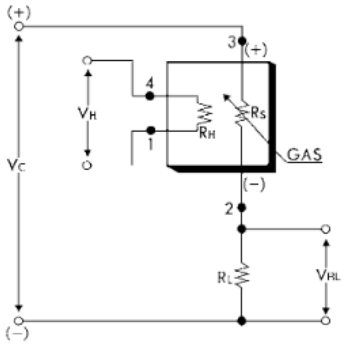


Figura 2.2: Imagen del sensor TGS2600 a la izquierda y esquema del sensor TGS2600, junto con el divisor de tensión utilizado para medir voltajes a la derecha. Figuras obtenidas de (Ortega, 2017).

con él. Se puede ver que tiene una carcasa exterior que lo protege de la suciedad del ambiente. En la imagen de la derecha se muestra un esquema del sensor.

El sensor tiene además dos resistencias cuyas salidas son los pines que se muestran en las imágenes. Una de ellas es la resistencia de calentamiento, R_H, que es la que regula la temperatura del sensor. Mediante el voltaje que se aplica en los pines de esta resistencia es posible variar la temperatura interna del sensor. El voltaje aplicado a estos pines se denomina voltaje de calentamiento, V_H. La resistencia marcada como R_S es la resistencia interna del sensor que varia su valor según los cambios producidos por el odorante en el sensor. La resistencia marcada con R_L es una resistencia de carga, que actúa como un divisor de tensión utilizado para medir el valor de la resistencia interna variable, R_S, a partir de la caída de potencial, V_{out} o V_{HL}, producida en la resistencia R_L, utilizando para ello la placa controladora. El voltaje que se obtiene de la caída de potencial sigue la fórmula 2.1.

$$V_{out} = ((V_C * R_L)/(R_S + R_L)) \tag{2.1}$$

De la cual resulta que el valor de la resistencia variable, R_S, se obtiene mediante la fórmula 2.2.

$$R_S = ((V_C * R_L)/V_{out}) - R_L \tag{2.2}$$

El valor del voltaje obtenido de la respuesta generada del sensor varía según la temperatura que se le aplique a este, siendo un factor importante a la hora de captar los odorantes. La secuencia de datos que se obtienen del sensor sobre un odorante se denomina: curva de respuesta del sensor.

2.3.1. Funcionamiento sensor TGS2600

Para captar la señal que produce un determinado odorante, o una mezcla de estos, es necesario que se genere una señal eléctrica en el sensor. El proceso de generación de la señal eléctrica en el sensor es el siguiente:

- El sensor, cuando está en presencia de aire no contaminante, genera una resistividad muy alta impidiendo que la corriente lo atraviese. Esto se produce porque el oxigeno se acumula en la superficie del sensor impidiendo que los electrones puedan circular libremente, ya que se ven atraídos hacia el oxígeno. Este proceso se muestra en la imagen 2.3 y 2.4.

- Cuando se expone al sensor a un gas, se produce una reacción de oxidación en la superficie de este. En esta reacción, los electrones se ven desplazados por las partículas del gas entrante. Esto provoca que el potencial de barrera que generaban las partículas de oxígeno disminuya y provoque un aumento en el flujo de electrones, que dependiendo de la composición química de la superficie del sensor y la temperatura a la que este se encuentre, dará una respuesta diferente. Este efecto se muestra en las figuras 2.3 y 2.4.

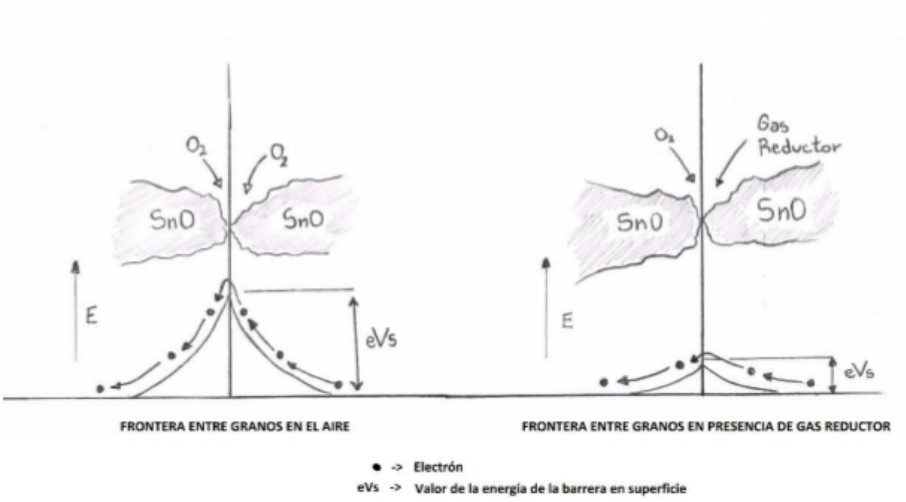


Figura 2.3: Interacción del sensor con el ambiente. A la izquierda en aire libre, a la derecha en presencia de un gas contaminante. Imagen adaptada de (Figaro, 2016a) obtenida de (Cruz Gutiérrez, 2016)

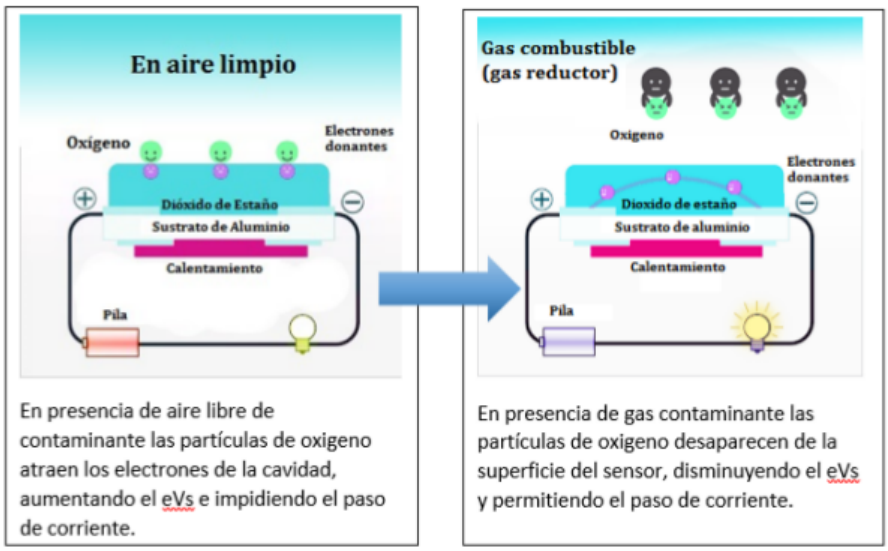


Figura 2.4: Comportamiento del sensor. Imagen adaptada de (Figaro, 2016b) obtenida de (Cruz Gutiérrez, 2016)

Como consecuencia de esta variación en la resistencia interna del sensor, se puede usar esta propiedad para la obtención directa de información de los odorantes que se analicen y su composición.

Durante la exposición del sensor a un odorante se produce una fase transitoria donde varia el valor de la resistencia interna del sensor del estado inicial, provocado por el odorante al que se

encuentra expuesto el sensor, hasta la saturación del valor de la resistencia. Cuando el sensor se ha saturado, la señal de la conductancia/resistencia se estabiliza entrando en una fase estable, ver figura 2.5.

Con saturación hacemos referencia a la estabilización de la señal de la resistencia/conductancia, debido a la llegada a un estado estable de la señal, produciéndose a diferentes valores dependiendo del odorante al que se exponga al sensor.

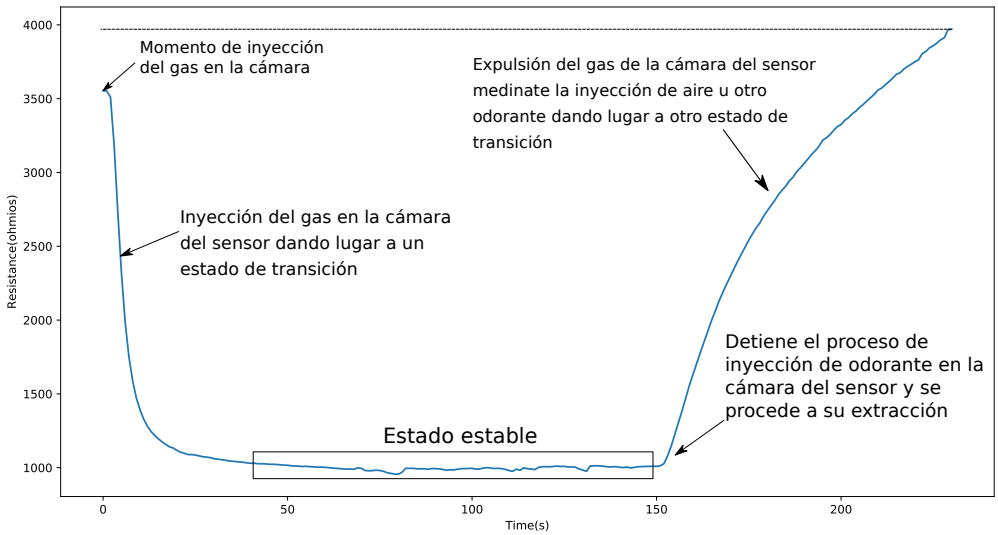


Figura 2.5: Respuesta del sensor a la exposición de un odorante. Se pueden observar tres fases diferentes: la exposición del odorante al sensor, la saturación del sensor y la fase de limpieza donde el sensor vuelve al estado inicial. La fase estacionaria se produce cuando el sensor se ha saturado y las fases transitorias cuando el sensor es expuesto al odorante y cuando se limpia la cámara del odorante.

2.4. Software de experimentación

El software que acompaña a la plataforma es el otro pilar de este proyecto. Este software permite el control de la placa de forma simple, utilizando un pseudo-lenguaje desarrollado específicamente para la plataforma (Ortega, 2017). Mediante el software de control podemos ejecutar los diferentes tipos de modulación implementadas en la plataforma de una manera muy simple, utilizando un script de ejecución sin tener que conocer el funcionamiento del software de la plataforma, aunque sí teniendo que tener un conocimiento básico de los elementos que componen la plataforma.

El software inicial se compone de un programa de ejecución llamado PyHuele y de tres módulos software. Cada uno de estos módulos se encarga de controlar un aspecto de la recogida y tratamiento de los datos obtenidos. Los módulos que componen el software son los siguientes:

- **Módulo de captura de datos:** se encarga de recoger los datos que se van captando del sensor y de procesarlos según requiera el algoritmo, para después guardarlos en ficheros. Este módulo consta de 4 scripts de ejecución, uno para cada tipo de modulación implementada en la plataforma, que son 3, y en el restante se encuentra el código común a todas las modulaciones.

- **Módulo de recogida y tratamiento de datos:** se encarga de recoger y procesar la información que define la ejecución de los experimentos de captura de datos, que recibe mediante scripts de ejecución, para que todo este correcto. Consta de un solo fichero con todas las rutinas de tratamiento de las cadenas de los scripts de ejecución.
- **Módulo de visualización de datos:** permite la representación gráfica de los datos obtenidos. Consta de un fichero con todos los métodos para representar gráficamente los datos.

Para realizar una captura de datos se pasa al programa principal un script de ejecución con todas las instrucciones para realizar la captura de los datos. El módulo de recogida y tratamiento de datos lee este script y realiza el tratamiento necesario. Tras el tratamiento de los datos del script, el programa principal PyHuele, selecciona el algoritmo de captura escogido y procede a la recogida de los datos. Por último, si queremos representar los datos captados, utilizamos los métodos de representación gráfica.

En el software de experimentación también está implementado un pseudo-lenguaje de experimentación. Este pseudo-lenguaje permite la realización de experimentos de captura de forma simple, ya que consta de una sintaxis sencilla, permitiendo la creación de scripts de ejecución complejos de forma simple. El pseudo-lenguaje está compuesto de la sintaxis y todas las palabras claves reservadas en él. Esto se verá con más detalle en el capítulo 3.

2.5. Técnicas de modulación

Como se comentó anteriormente, la plataforma consta de varios circuitos que permiten realizar capturas con diferentes tipos de modulaciones, ver sección 2.2. Las modulaciones se diferencian según el modo en que se varia la temperatura que se le aplica al sensor. Esta temperatura se regula mediante el voltaje de de calentamiento, V_H, como se explicó anteriormente en la sección 2.3.1. A través del término modulación hacemos referencia a la variación de la señal de temperatura a partir de los datos previamente captados. Mediante la variación de esta temperatura se busca cambiar las propiedades físicas del sensor, variando las reacciones químicas que ocurren en su superficie con la intención de obtener más información que manteniendo la temperatura constante para un mismo odorante.

2.5.1. Adquisición de odorantes a temperatura constante

Este modo de adquisición de odorantes es el más simple de todos los implementados. Consiste en fijar para un valor determinado la temperatura del sensor al inicio del proceso de captura, y no variarla de ninguna manera. Este modo de captura es el más utilizado actualmente a la hora de medir odorantes.

Una ligera variación de este modo de adquisición de datos consiste en fijar la señal de la temperatura durante un tiempo, para luego variar su amplitud de tal forma que en la señal de la temperatura se formen *escalones* (Hossein-Babaei and Amini, 2014). De este modo se produce una variación en la señal inducida por este choque térmico con la intención de que la huella obtenida ayude a diferenciar los odorantes que se analicen (Hossein-Babaei and Amini, 2012).

2.5.2. Adquisición de odorantes mediante temperatura variable: codificación en amplitud

En este modo de adquisición de odorantes se ajusta la temperatura del sensor mediante los valores obtenidos previamente de este. Mediante esta iteración entre el voltaje obtenido del

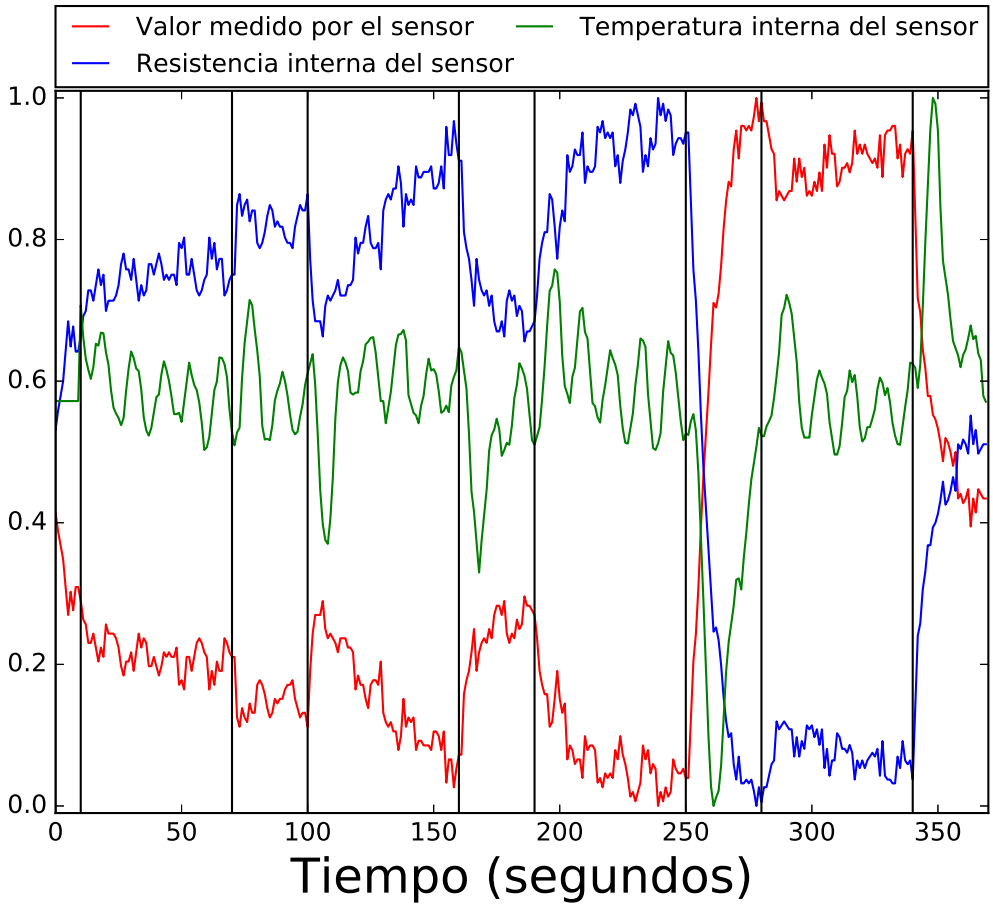


Figura 2.6: Gráfica de una captura siguiendo el temperatura variable y codificación en amplitud. La secuencia de gases es: muestras iniciales, aire, metanol, aire, etanol, aire, butanol y aire. Cada linea vertical representa un cambio de gas. En la figura, cada una de las señales se ha normalizado entre 0 y 1, ya que de otra forma no podían representarse juntas, pudiendo así visualizar las 3 señales conjuntamente.

sensor y su temperatura, se varía dinámicamente el voltaje de calentamiento que se le aplica al sensor consiguiendo una huella de temperatura característica para cada gas (Gutierrez-Osuna et al., 2003; Vergara et al., 2005; Gosangi and Gutierrez-Osuna, 2013).

Mediante esta auto-regulación, realizada a través de la aplicación de una regresión lineal a la señal del voltaje obtenida del sensor, se consigue que la señal que se va obteniendo no llegue a rangos superiores a los permitidos por la placa controladora. Cuando la señal del voltaje se encuentra cerca de estos valores, el algoritmo regula la temperatura que se le aplica al sensor, evitando la perdida información. En la imagen 2.6 se muestra un ejemplo del funcionamiento de este algoritmo. En esta figura se puede observar como las señales del voltaje medido, la señal roja, y del voltaje que se le aplica al sensor, la señal verde, son opuestas con un cierto desfase. Este desfase es debido a la aplicación del algoritmo, ya que el cálculo del voltaje que regula la temperatura no es instantáneo, sino que se requieren unos pocos datos para ver la evolución del voltaje. Este efecto se observa en la transición que comienza en el segundo 250, en la figura 2.6, donde el voltaje comienza a aumentar con la inyección del nuevo odorante, y para evitar que la señal llegue al límite permitido por la placa controladora, se va disminuyendo progresivamente la temperatura de calentamiento del sensor, compensando así la subida del voltaje captado. En la última transición se puede ver el efecto contrario.

Cada nuevo valor de la temperatura se calcula sobre una ventana de medidas del voltaje

obtenido, a través de la aplicación de una regresión lineal a estos datos. De este modo, la temperatura que se aplica al sensor se regula con cada nueva captura, recalculando la temperatura que se le va a aplicar al sensor mediante la fórmula 2.3.

$$T_{heat\_new} = T_{heat\_old} - (SLOPE * TEDENCIA) \tag{2.3}$$

En ella, el término SLOPE hace referencia a la pendiente que se obtiene al aplicar una regresión lineal a los datos obtenidos del sensor. Esta regresión lineal no se aplica sobre el conjunto total de los datos, sino que se utiliza una ventana de valores. El parámetro TENDENCIA es un multiplicador que se le pasa al algoritmo y que fija la variación de la amplitud en el nuevo valor de la temperatura.

El valor que se utilice como ventana tiene una gran importancia, ya que esta ventana define el total de datos que se van a utilizar para aplicar la regresión. Si se utiliza una ventana de datos pequeña, la regresión no tendrá suficiente "memoria" de los datos capturados, y presentará un comportamiento errático, dando lugar a una señal ruidosa. Para ventanas grandes se puede llegar a utilizar transiciones completas en el cálculo de la regresión, reduciendo así la eficacia del algoritmo. El valor de la tendencia utilizada también es de gran importancia, ya que define la amplitud de la variación de la señal. Si se utiliza un valor pequeño, la amplitud de la señal apenas variará, mientras que si se utiliza un valor muy grande, la señal se encontrará dando bandazos entre el mínimo y el máximo permitidos.

2.5.3. Adquisición de odorantes mediante temperatura variable: codificación en frecuencia

Este método de adquisición de odorantes se basa en la premisa de que la respuesta que registra un sensor de un determinado odorante es diferente según la temperatura a la que se encuentra el odorante, teniendo cada odorante una sensibilidad distinta según la temperatura a la que se encuentre (Martinelli et al., 2012, 2013a).

Este método utiliza un circuito cerrado, en el cual la salida del sistema sirve como retroalimentación a la entrada del sensor, permitiendo ajustar la temperatura del sensor en base a los datos captados, ver figura 2.7. El circuito cerrado se compone de dos bloques, el primero está formado por el sensor y una interfaz electrónica cuyo objetivo consiste en captar y procesar la señal obtenida. El segundo bloque está compuesto por una interfaz que procesa la señal de salida, Xout en la figura 2.7, para obtener la nueva temperatura de calentamiento.

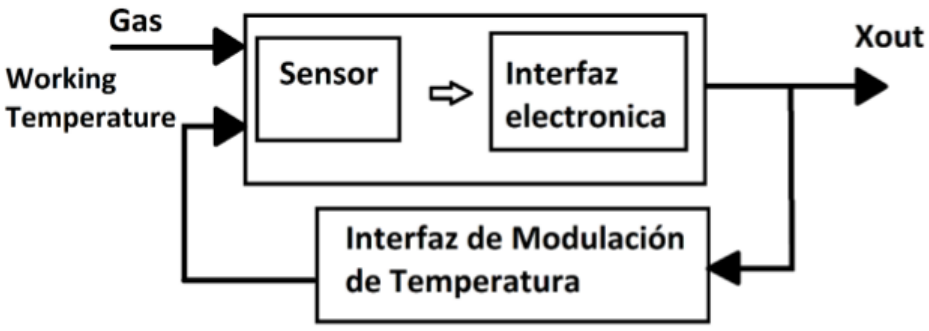


Figura 2.7: Diagrama de bloques propuesto para la adquisición de odorantes mediante temperatura variable y codificación en frecuencia. Figura adaptada de (Martinelli et al., 2012).

El circuito que se compone de un multivibrador astable. Este componente genera una señal periódica que varia entre dos estados. La frecuencia con la que se genera esta señal periódica

se ve afectada por la resistencia del sensor a un odorante. Según el odorante que se analice se variará la resistencia interna del sensor, que afectará en mayor o menor medida la frecuencia con la que se generan estos pulsos.

Cuanto mayor sea la resistencia que genere el odorante en el sensor, el circuito generará unos pulsos más amplios, por lo que las oscilaciones durarán más tiempo. Si del sensor se obtiene una resistencia baja, la oscilación generada durará menos tiempo, generando un alto número de pulsos. La generación de estos pulsos también se ve afectada por el modo de variar la temperatura que se utilice.

La interfaz de modulación térmica se compone de un contador software, que tras registrar un conjunto de oscilaciones, se produce una transición entre los estados lógicos definidos. En la imagen 2.8 se muestra la señal obtenida del multivibrador astable en verde, la señal de la temperatura que se aplica al sensor en rojo y el contador software en azul.

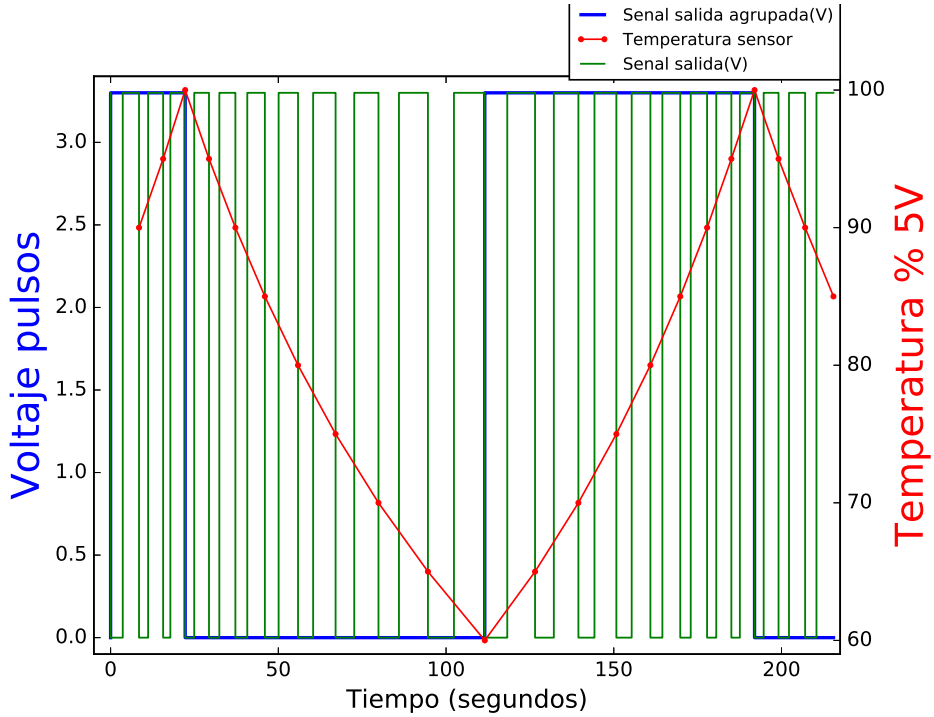


Figura 2.8: Representación de los pulsos producidos por el circuito, junto con la transición entre ambos estados y la temperatura del sensor. Imagen obtenida de las capturas de la plataforma generadas durante la realización de (Ortega, 2017).

2.5.4. Adquisición de odorantes mediante temperatura variable: codificación con PID

Mediante el uso de esta técnica de adquisición se busca descubrir los valores de conductancia máximos y mínimos característicos de cada gas, a la vez que se exploran los diferentes valores de conductividad.

Para explorar el comportamiento del sensor en diferentes estados, se busca variar la respuesta del sensor de manera oscilatoria. Para conseguirlo, en vez de variar la señal de la temperatura (Nakata et al., 1996), se utiliza una señal *target* de voltaje sinusoidal y se varia el perfil de temperatura para conseguir que la respuesta que se obtiene del sensor siga esta señal. El periodo utilizado para la generación de la señal sinusoidal afectará en gran medida al resultado obtenido (Velasco Botina, 2017).

Esta señal de referencia se va actualizando utilizando para ello los datos obtenidos del sensor. Mediante esta variación de la señal se consigue explorar el comportamiento del sensor en diferentes estados. Para la actualización de la señal de referencia se utiliza un sistema PID (Herrero-Carrón et al., 2015), ver figura 2.9.

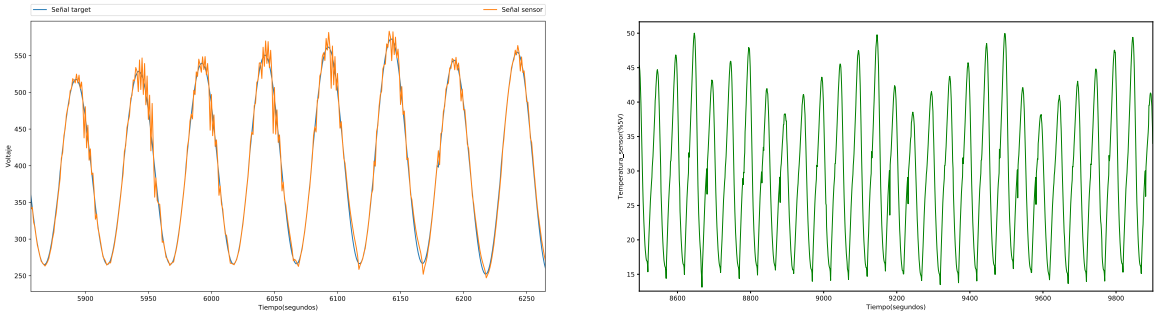


Figura 2.9: La figura de la izquierda muestra una captura utilizando el algoritmo de codificación con PID. La señal azul representa el *target* que debe seguir el sensor y la señal naranja representa el valor del sensor. La gráfica de la derecha muestra el valor de la temperatura que se aplica en el sensor para poder seguir la señal de objetivo.

Un controlador PID es un sistema retro-alimentado que calcula la diferencia entre una señal de referencia y el valor obtenido de un sistema de medición. Está compuesto por 3 componentes:

- Componente proporcional(K_p): depende de la diferencia entre la señal de referencia y la señal obtenida.
- Componente integral(K_i): depende de los errores cometidos pasados, que se van acumulando.
- Componente derivativa(K_d): depende de la tasa de cambio de la señal obtenida, midiendo los posibles errores futuros.

Este sistema de control es un sistema en lazo cerrado, muy utilizado en sistema de control industrial y que se ha utilizado en la captura de datos de odorantes. Para el funcionamiento de este mecanismo se necesitan los siguientes elementos:

- Un sensor para obtener o determinar el estado del sistema.
- Un actuador que se encargue de modificar al sistema de forma controlada.
- Un controlador que genere la señal que gobierne al actuador.

El sensor proporciona información sobre el estado actual en el que se encuentra el sistema. El controlador genera una señal, cuyo rango es el mismo que la señal proporcionada por el sensor, que es el objetivo al que se quiere llegar. Esta señal generada se denomina punto de referencia o setpoint. Ambas señales se restan proporcionando una señal de error, que es la que determina la diferencia entre el valor deseado y el medido. Esta señal es utilizada por cada una de las diferentes componentes, siendo la suma de todas las componentes la salida final que gobierna al actuador.

2.6. Tratamiento de las señales

Una vez que se hayan obtenido varias señales sobre diferentes odorantes, y antes de aplicar técnicas de aprendizaje automático para clasificar los datos, se han implementado y aplicado varias técnicas con las que se pretende corregir la deriva inherente al sensor y eliminar interferencias exteriores que afectan a la señal como pueden la temperatura y humedad exterior. A continuación se procede a explicar las técnicas utilizadas en el contexto del problema que rodea a las narices electrónicas.

2.6.1. Transferencia de calibración

Debido a los tratamientos químicos a los que son sometidos los sensores durante el proceso de fabricación, la composición química varía entre ellos. Esto unido a la contaminación del sensor por medio de un uso continuado a diferentes odorantes a los que se es expuesto, las variaciones provocadas por los efectos ambientales y al volumen de odorante expuesto, entre otros, provocan que las medidas que se obtienen muestren una diferencia y una degradación con el tiempo de los valores obtenidos, aún utilizando el mismo sensor y la misma concentración en los odorantes utilizados, tal y como se comenta en la sección 1.1.

Mediante el uso de técnicas de calibración se pretende mapear el espacio de señales de un sensor, denominado maestro o master, a otro sensor, denominado esclavo o slave, un conjunto de sensores o incluso a sí mismo para corregir los efectos del drift mediante el uso de un conjunto de muestras de transferencia (Fonollosa et al., 2016). Las muestras de transferencia son un conjunto de datos representativos utilizados para eliminar las diferencias presentes en las señales, realizando con ellas el mapeo de las señales obtenidas de los diferentes sensores que se han utilizado. Estas muestras se componen de las diferentes características obtenidas para un sensor, como puede ser el voltaje registrado o la temperatura de calentamiento, característicos para un sensor, de tal forma que una muestra de transferencia puede estar compuesta de diferentes valores característicos de un sensor.

Mediante la aplicación de estas técnicas se obtiene una linea base con la que se puede operar, ya que aunque sigan existiendo variaciones en los datos obtenidos, se han mitigado estos efectos de tal forma que ya no son tan evidentes (Fernandez et al., 2016).

Este tipo de técnicas se utilizan mayoritariamente en el ámbito de la espectrometría para la calibración de instrumentos infrarrojos (Ji et al., 2015; Bouveresse and Massart, 1996; Liu et al., 2014), adaptándolas para su uso en el campo de las narices electrónicas.

Las estrategias de calibración se dividen en dos grupos diferentes (Wise and Roginski, 2015):

- El primer grupo está formado por los métodos que tratan de mapear el espacio de señales adquiridas al espacio de señales utilizadas como maestras.
- El segundo grupo lo forman las metodologías cuyo objetivo es la eliminación de la variación de las respuestas en instrumentos diferentes.

Los métodos de calibración más utilizados son los siguientes:

- **Direct Standardization (DS):** esta técnica consiste en el mapeo del espacio de muestras maestras mediante la transformación lineal:

$$S_M = S_S * F.$$

Donde S_M y S_S corresponden con las matrices de los datos utilizados como maestro y esclavo respectivamente, formadas por las muestras de transferencias seleccionadas y utilizadas para el mapeo de las señales obtenidas. F se corresponde con la matriz de transformación que permite el mapeo de las nuevas muestras obtenidas al espacio de datos utilizado como maestro. Esta matriz se obtiene mediante la matriz pseudo-inversa de S_S:

$$F = (S_S)^{-1} * S_M.$$

Por último, el mapeo de una muestra al espacio del maestro se realiza aplicando la fórmula 2.4. El vector x contiene los datos que queremos ajustar, F es la matriz de transformación y x_F es el vector de datos una vez que se le ha aplicado la transformación:

$$(x_F)^T = x^T * F. \tag{2.4}$$

Esta técnica será utilizada en los análisis posteriores realizados a los datos como se muestra en el capítulo 6. Se ha decidido el uso de esta técnica debido a la facilidad del cálculo de la matriz de transformación F y a la realización del mapeo de las muestras, así como los buenos resultados que aporta.

- **Piece-wise Direct Standardization (PDS):** PDS se utiliza cuando el número de características de una muestra es mayor al número al número de muestras (Wang et al., 1992). Para evitar este problema se utilizan modelos lineales, f_j, que relacionan la respuesta del sensor maestro con las del sensor esclavo, utilizando para ello ventanas de tamaño reducido en las diversas variables que confortan el espacio de muestras del maestro y del esclavo. La matriz F resultante tiene una estructura diagonal:

$$F = diag(f_1)^T, (f_2)^T, (f_3)^T, ..., (f_k)^T).$$

Para realizar el mapeo de los espacios se utiliza la formula 2.4.

- **Orthogonal Signal Correction (OSC):** trata de suprimir aquellas fuentes de ruido o varianza presentes en los datos de los instrumentos esclavos (X) que son ortogonales a los datos maestros (Y) (Fearn, 2000). Para ello se busca aplicar a los datos un vector de pesos que elimine estas fuentes de variación. Este vector de pesos, w, es el autovector obtenido de la matriz: MXX^T cuyo autovalor sea el mayor. La matriz M es el resultado de aplicar la fórmula 2.5.

$$M = I - X^T Y (Y^T Y X X^T)^{-1} Y^T X. \tag{2.5}$$

Donde I representa la matriz identidad. Una vez obtenido el vector w, solo hay que aplicarlo al conjunto de datos deseado para eliminar las fuentes de varianza.

- **Generalización de mínimos cuadrados ponderados (GLSW):** este método identifica y pondera negativamente aquellas características que son responsables de una mayor fuente de varianza en los datos obtenidos del sensor mediante la aplicación de un filtro. Para construir este filtro se calcula una matriz, denominada C, a partir de la diferencia entre las matrices de los datos maestro y esclavo, generadas a partir de las muestras de transferencia, centrando ambas matrices en la media:

$$C = (S_M - S_S)^T * (S_M - S_S).$$

Después, se factoriza C, descomponiéndola en dos matrices diferentes a partir del teorema de la descomposición singular de valores, obteniendo las matrices de autovectores (V) y la matriz de valores singulares (D):

$$C = V * D^2 * V^T.$$

Tras factorizar la matriz C, se calcula la matriz de pesos de los autovalores utilizando la fórmula 2.6, que es con la que se calculará la matriz filtro.

$$W = \sqrt{(D^2/\alpha) + I}. \tag{2.6}$$

La incógnita I representa la matriz identidad y α es un parámetro de pesado, que controla el grado de similitud de los instrumentos maestro y esclavo. Con un valor alto, incrementamos la ponderación negativa que se le aplica a las características que son causantes de la mayor fuente de varianza y viceversa. La matriz filtro utilizada en la eliminación de las características que más ruido aportan, se calcula utilizando la fórmula 2.7.

$$G = V * W^{-1} * V^T. \tag{2.7}$$

Por último, solo hay que aplicar esta matriz filtro a los datos para ponderar negativamente los efectos provocados por las características que aportan varianza a los datos.

2.6.2. Decorrelación de la humedad y la temperatura

Los sensores MOS que se utilizan en la plataforma son muy sensibles a las variaciones de la temperatura y la humedad ambientales (Morante, 2013; Barsan and Weimar, 2003). Estas variaciones no afectan de la misma manera a todos los sensores, pudiendo mitigar estos efectos mediante el uso de diversas técnicas de filtrado (Huerta et al., 2016), obteniendo así una respuesta más limpia y facilitando la clasificación y detección de eventos (Rodriguez-Lujan et al., 2013), o por el propio diseño del sensor.

Para la supresión de los efectos de la temperatura y la humedad ambientales, mediante el uso de filtros, se utilizan modelos basados en las bandas de energía de los semiconductores. Una banda de energía cuantifica la variación de la resistencia del sensor durante la exposición de un odorante a este (Morante, 2013; Barsan and Weimar, 2003). Estas variaciones en la resistencia del sensor son las que regulan todo el proceso de transformación, denominado transducción, de la señal química en una señal eléctrica viéndose afectado por los cambios producidos por las condiciones externas.

El modelo utilizado consta de tres parámetros libres que se ajustan a los datos de entrenamiento: β_1, β_2 y β_3. Para ajustar el modelo a los datos de entrenamiento se utiliza la fórmula 2.8.

$$ln(R_F/R_I) = \beta_1 * \Delta H + \beta_2 * \Delta H^2 + \beta_3 * \Delta H * \Delta T_E. \tag{2.8}$$

Donde R_I y R_F son las resistencias del sensor en los instantes de tiempo $t-1$ y t respectivamente. ΔH y ΔT_E se corresponden con las variaciones de la humedad y la temperatura ambiental, durante el periodo de tiempo implicado en el empleo de esta técnica. Estas variaciones de los datos se corresponden con la diferencia de la humedad y la temperatura ambiental en tiempo t menos la humedad o temperatura ambiental en tiempo *t-1*. Este método se ha ajustado a una base de datos mediante el uso de una regresión lineal como se muestra en el capítulo 6.

2.7. Obtención de características de las señales

De las curvas de respuesta del sensor obtenidas se extraerán diferentes características para su clasificación posterior. La extracción de características consiste en mapear el espacio original de características a un espacio de menor dimensión preservando la mayor cantidad de información

de la señal original (Pardo et al., 1998). Mediante esta reducción de la dimensionalidad podemos clasificar los datos más rápido intentando que el resultado sea lo más fidedigno posible utilizando para ello varias técnicas de obtención de características.

2.7.1. Exponential Movement Average

La idea en la que se basa este modo de extracción de características es que en la fase transitoria que se da durante la variación de la señal se puede extraer más información que durante la fase estacionaria. Como se comentó anteriormente, hay dos formas de variar la señal obtenida del sensor:

- Variación en la concentración analizada (Martinelli et al., 2003; Wilson and DeWeerth, 1995).
- Variación en la temperatura del sensor (Cavicchi et al., 1996; Vergara et al., 2007).

Esta técnica proviene del campo de la econometría y fue adaptada para el campo de las narices artificiales (Muezzinoglu et al., 2009; Vergara et al., 2012, 2010). Esta técnica es un indicador ampliamente utilizado que ayuda a suavizar el ruido de la señal de fluctuaciones presentes en ella, dando más peso a los datos más recientes. Mediante el uso de esta técnica se busca obtener los valores de pico máximo y mínimo de la señal, a partir de la señal de resistencia del sensor. La resistencia del sensor se capta a una frecuencia determinada y se utiliza para calcular la señal EMA para un tiempo discreto utilizando la fórmula 2.9.

$$y[k] = (1 - \alpha) * y[k-1] + \alpha * (x[k] - x[k-1]). \tag{2.9}$$

El parámetro α es un parámetro de suavizado, cuyo valor está entre 0 y 1, modelando la calidad de la serie obtenida. El valor inicial de EMA es 0, y[0] = 0, y el valor de k hace referencia a cada valor de la resistencia obtenida del sensor, x. Como se puede observar en la fórmula 2.9, según el valor otorgado a α se suaviza más el último valor obtenido de la serie EMA, la parte izquierda de la ecuación, o se suaviza la diferencia de los puntos k y $k-1$ de la serie de valores de la resistencia del sensor que se encuentra en la parte derecha de la ecuación.

Las características que se extraerán de las señales y se utilizarán para clasificar los datos son el valor máximo y mínimo de la señal EMA, para unos valores α de: 0.1, 0.01, 0.001 (Vergara et al., 2012), ver figura 2.10. También se utilizarán características estacionarias de la señal, las cuales consisten en obtener la diferencia entre el valor máximo de la señal de la resistencia y el valor mínimo:

$$\Delta R = max(R) - min(R).$$

También se utilizará este valor normalizado:

$$||\Delta R|| = (max(R) - min(R))/min(R).$$

2.7.2. Obtención de características mediante el uso del bulbo olfativo

Este método implementado para su utilización en la obtención de características funciona en dos etapas (Jing et al., 2016). La primera etapa consiste en la transformación de las curvas de respuestas obtenidas de los sensores en *spikes*. Para realizar esta transformación se utiliza

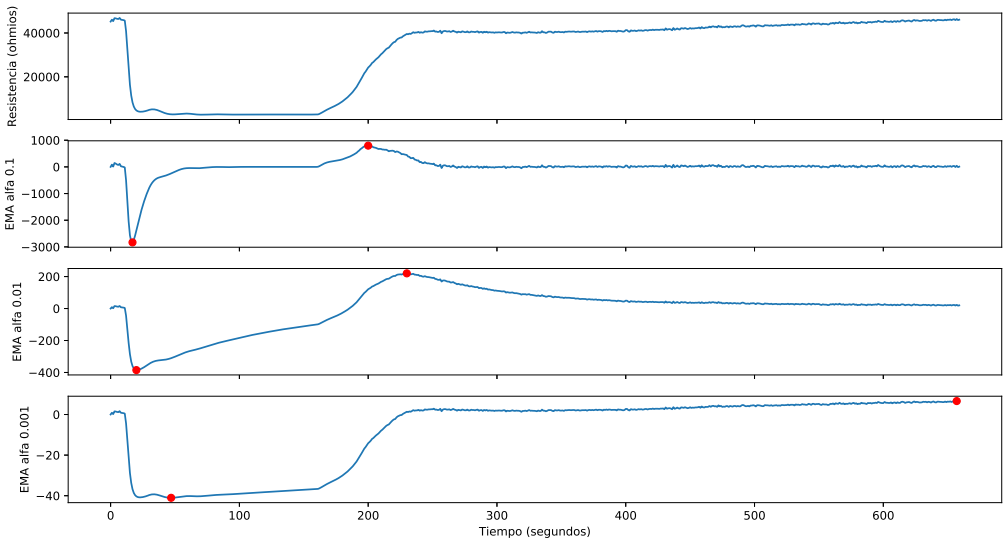


Figura 2.10: Obtención de EMA para diferentes valores de α. La primera imagen muestra la señal de la resistencia original, el resto los valores de EMA para diferentes valores de alfa, indicando los valores máximos y mínimos de cada señal en rojo.

una red neuronal que imita el sistema olfativo de ciertos mamíferos menores. Este mecanismo de extracción de características se basa en la premisa de que el sistema olfativo de los seres vivos se encuentra continuamente expuesto a diferentes odorantes en diferentes concentraciones, pudiendo transformar estos estímulos en actividad neuronal y diferenciarlas.

La red utilizada se muestra en la imagen 2.11. Esta red es una mejora de la utilizada en (Polese et al., 2014). Su estructura está compuesta por cinco neuronas:

- Receptores olfativos (ORN - Olfatory receptor neurons).
- Células mitrales (MC - Mitrals Cells).
- Células granulares (GC - Granule Cells).
- Células periglomerulares (PC - Periglomerular Cells).
- Células de axón corto (SAC - Short Axon Cells).

La entrada a la red la forman las células ORN. Después, la señal pasa a las células mitrales y periglomerulares, que se encuentran conectadas de forma inhibitoria (Valova et al., 2007). La información pasa por el ciclo formado por las células mitrales y granulares, siguiendo el siguiente orden: mitral, granular y mitral. Este circuito inhibitorio puede mejorar las señales espacio temporales, generadas a partir de las curvas de respuesta del sensor a los diferentes odorantes, desacoplándolas para diferenciarlas del resto. Las células de axón corto permiten la comunicación entre las diferentes células mitrales, permitiendo la dispersión de la información mediante conexiones excitatorias. Las conexiones marcadas con un "+" representan conexiones excitatorias y las conexiones marcadas con un "-" representan conexiones inhibitorias. La salida de la red la forman la salida de las células mitrales. La transformación que sufren las curvas de respuesta del sensor se muestran en la figura 2.12.

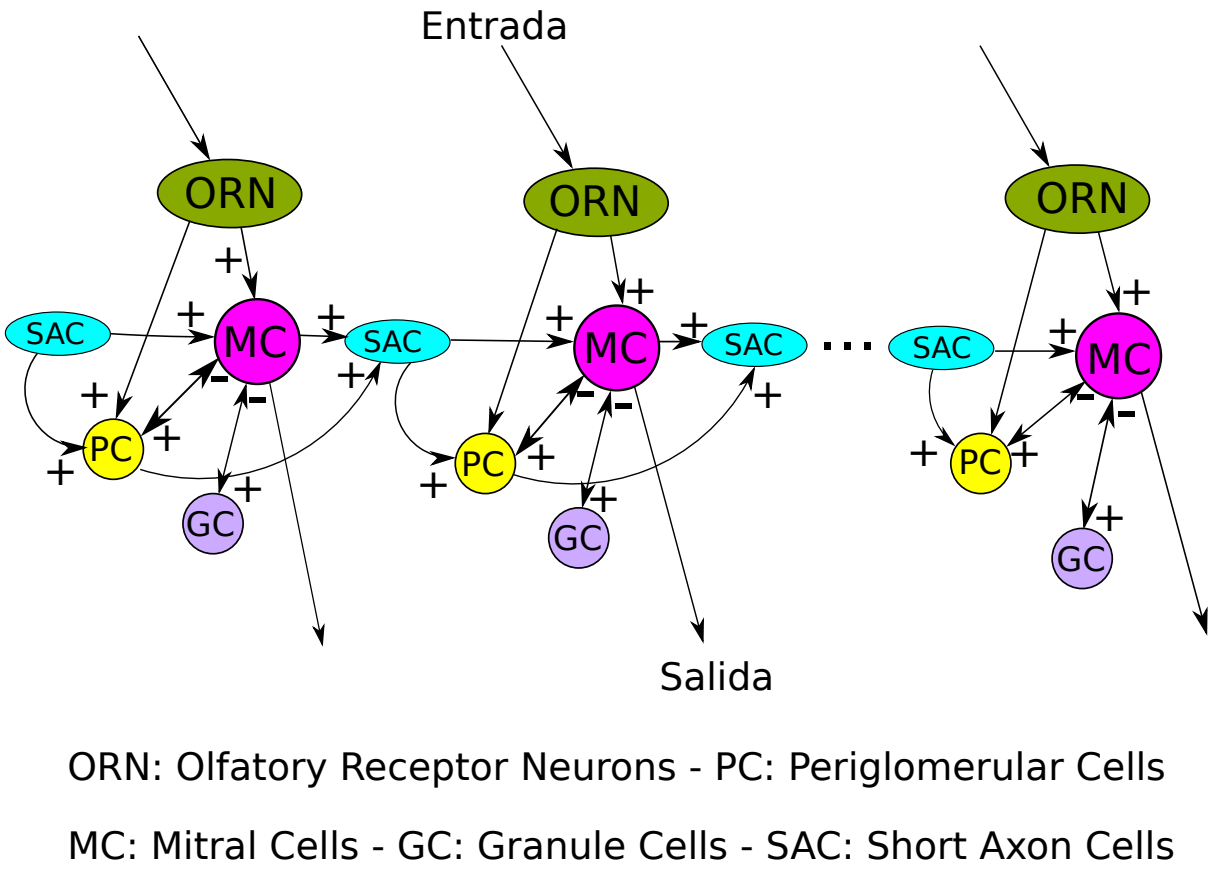


Figura 2.11: Imagen de la red utilizada para la transformación de las curvas de respuesta del sensor. Imagen obtenida de (Jing et al., 2016).

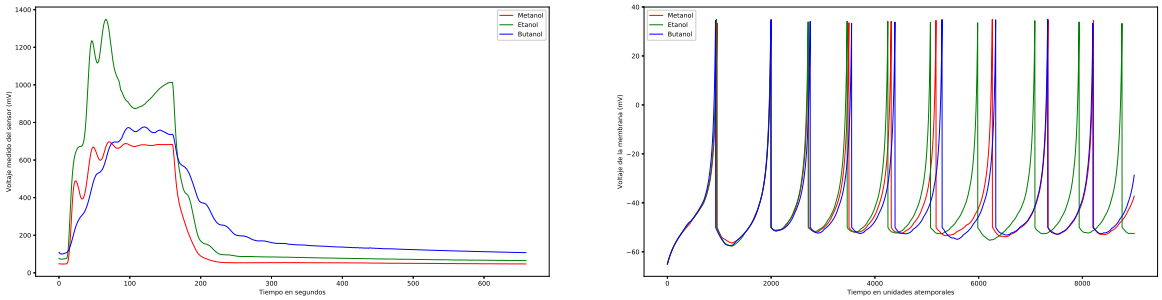


Figura 2.12: Transformación de la curva de respuesta del sensor en actividad neuronal. La imagen de la izquierda muestran tres curvas para diferentes odorantes: metanol, etanol y butanol para diferentes capturas realizadas utilizando la plataforma del GNB. La imagen de la derecha muestra la actividad neuronal obtenida de la red a partir de las curvas de odorantes previas. La salida es casi la misma debido a que las señales no se han podido desacoplar debidamente; aun así puede verse una similitud entre las señales roja y azul, tanto en la imagen de la izquierda como en la derecha, mientras que la señal verde difiere más de las series mostradas en la imagen de la izquierda, quedando esto reflejado en el desfase que se aprecia en la imagen de la derecha.

Para realizar la simulación de la salida que produce esta red neuronal se va a simular la actividad neuronal del circuito, utilizando para ello modelos neuronales. Estos modelos simulan el potencial de membrana de una neurona. Este potencial se ve afectado por las conexiones que mantienen las neuronas entre sí, que pueden ser tanto excitadoras como inhibidoras. Cuando

el potencial de membrana alcanza su valor máximo se produce un *spike*, para después volver a su estado inicial. Mediante la generación de estos *spikes* las neuronas son capaces de codificar la información y los estímulos que reciben del exterior, transformándolos en impulsos eléctricos para su procesamiento. En la actualidad hay varios modelos neuronales que simulan el potencial de membrana de una neurona con mayor o menor precisión siendo los modelos más comunes (Izhikevich, 2003; Hodgkin and Huxley, 1952; Hindmarsh and Rose, 1984)

La segunda parte de este método consiste en la extracción de características de las señales. Debido a que la respuesta obtenida de la red no es estacionaria se utilizarán diagramas de recurrencia o Recurrence Plot (RP) y Recurrence Quantification Analysis (RQA) (Zbilut and Webber Jr, 1992).

Los diagramas de recurrencia son técnicas utilizadas para el análisis de sistemas no lineales. Se visualizan mediante un grafo o una matriz cuadrada, donde cada elemento que constituye ese grafo o matriz se corresponde a un estado del sistema dinámico. Para obtener aquellos puntos donde se repite un estado del sistema dinámico se utiliza el espacio de fases del sistema, observando cuando la trayectoria del espacio de fases visita de forma aproximada una zona que previamente ya ha visitado.

Para realizar el diagrama de recurrencia, se discretiza el espacio de fases, contabilizando las recurrencias como cualquier punto donde la trayectoria del espacio de fases se acerca lo suficiente a un punto donde ha estado previamente, denotando esta cantidad como ϵ. Estas recurrencias se pueden medir mediante la fórmula binaria 2.10. En la figura 2.13 se muestra un recurrence plot.

$$R(i,j) = \left\{ \begin{array}{l} 1 \text{ si } ||x(i) - x(j)|| <= \epsilon \\ 0 \text{ en otro caso} \end{array} \right\} \tag{2.10}$$

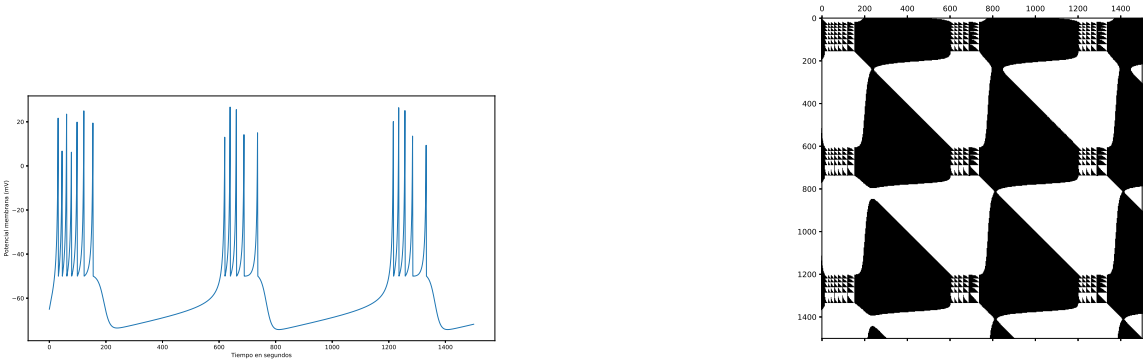


Figura 2.13: Esta figura muestra el potencial de membrana obtenido aplicando el modelo de Izhikevick (Izhikevich, 2003) y el RP obtenido con un umbral de 0.05. Los valores en negro se corresponden con recurrencias del espacio de fases.

Tras calcular los RP, el siguiente paso es contabilizar las longitudes de los conjuntos de unos obtenidos de la ecuación 2.10 que se encuentren seguidos, tanto en diagonal como en vertical, para poder aplicar las técnicas RQA.

Las técnicas RQA se desarrollan para cuantificar los gráficos de recurrencia (Recurrence Plots). Para ello se basan en las lineas diagonales y verticales. Al ser los RP simétricos, las lineas horizontales son iguales a las verticales, utilizando esta últimas. Estas lineas se corresponden con el comportamiento del sistema en el espacio de fases: las lineas diagonales muestran segmentos de la trayectoria del espacio de fases que se ejecutan en paralelo, mientras que las

lineas verticales representan segmentos que permanecen en una misma región durante un tiempo. Las características más comunes que se utilizan, y que serán explicadas con más detalle en la sección 4.2.5.2, sobre los RP para extraer información son:

- Ratio de recurrencia (Recurrence Rate): mide la densidad de puntos de recurrencia para un sistema dinámico. Contabiliza la probabilidad de que un determinado estado se repita.
- Determinismo (Determinism): contabiliza el número de puntos de recurrencia que forman líneas diagonales y están relacionadas con la previsibilidad del sistema.
- Laminaridad (Laminarity): contabiliza el número de puntos de recurrencia que forman líneas verticales. Está relacionado con la intermitencia en la ocurrencia de los estados.
- Tiempo de atrapamiento (Trapping Time): mide la duración de un sistema dinámico en un determinado estado. Está relacionado con la laminaridad.
- Longitud de la línea diagonal promediada (Averaged diagonal line length): mide la previsibilidad del sistema.
- Entropía de las lineas diagonales y verticales (Entropy of diagonal and vertical lines): miden la complejidad de la estructura determinista del sistema.
- Longitud de las lineas verticales y diagonales más largas (Length of the largest vertical and diagonal lines): miden la longitud de los segmentos que corren paralelos en el espacio de fases.

2.7.3. Señales puras

Además del uso de las técnicas de extracción de características aplicadas a las señales que se vayan a clasificar, se utilizará la señal completa. Esta señal, obtenida de las capturas del sensor, se dividirá según las transiciones realizadas, y sobre ella se aplicarán directamente técnicas de aprendizaje automático. El vector de características estará formado por la longitud de la transición de un odorante a otro.

De este modo se pretende comprobar la eficacia de las técnicas de extracción de características, examinando si los resultados son satisfactorios, o por el contrario, no merece la pena utilizar un método de extracción de características para clasificar un conjunto de datos.

3

Sistema y diseño

En este capítulo se procede a explicar como se han diseñado y estructurado el sistema software de la plataforma, y las técnicas utilizadas en el tratamiento de los datos capturados. La figura 3.1 muestra un esquema general de los elementos desarrollados durante la realización de este TFM, junto con el proceso de captura y clasificación de los datos. En el anexo B se muestra un manual para que un usuario pueda ejecutar experimentos fácilmente.

3.1. Software y pseudo-lenguaje de experimentación

A lo largo del desarrollo de este TFM se han realizado diversas modificaciones al software original de la plataforma para la ejecución de los experimentos, así como para su uso en otros trabajos paralelos de fin de máster relacionados con el ámbito de las narices artificiales (Velasco Botina, 2017).

Como se comento en la sección 2.4, el software de la plataforma se encuentra dividido en tres módulos. Cada uno de estos módulos se encarga de un aspecto necesario de la captura y procesamiento de los datos obtenidos, junto con su representación gráfica.

El software inicial ha sufrido diversas modificaciones, añadiéndole un conjunto de funcionalidades nuevas, con la intención de hacer a la plataforma más configurable. Con el nuevo software no solo se puede configurar los experimentos, sino también las características de la plataforma que inicialmente eran invariables, como la frecuencia de captura o la resistencia del divisor de tensión, entre otros posibles. Para poder configurar la plataforma como se desee, se hará uso de otro script de configuración, del mismo modo que el utilizado para configurar los experimentos. De este modo se pasa a tener dos scripts de configuración: uno para los experimentos y otro para la plataforma.

Además de los cambios anteriormente citados, también se han añadido un conjunto de técnicas para el análisis y procesamiento de las señales obtenidos del sensor, con el objetivo de clasificarlas.

El pseudo lenguaje también ha sufrido varias modificaciones acordes a la adición del nuevo tipo de modulación, explicada en la sección 2.5.4. Debido a la incorporación del algoritmo de modulación se han añadido nuevas palabras claves para configurar todos los aspectos variables

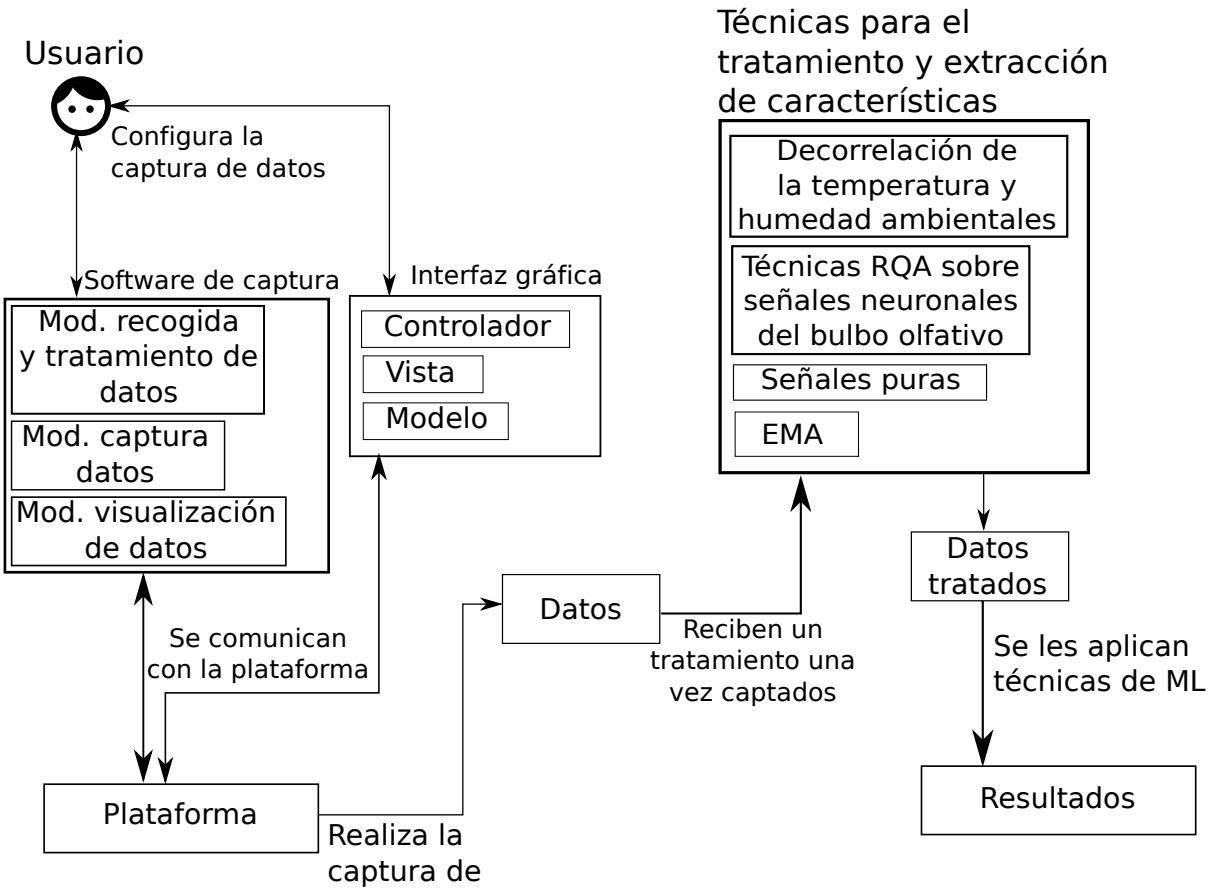


Figura 3.1: Imagen que representa el proceso de captura y clasificación de los datos, junto con todos los elementos implementados durante el desarrollado de este TFM.

de este algoritmo. Del mismo modo, se han añadido todas las palabras claves necesarias para realizar una configuración total de los diferentes elementos de la plataforma, durante la captura de los datos sin necesidad de estar modificando el código de la plataforma, sino mediante el uso del script que configura la plataforma.

Estas ampliaciones son muy importantes ya que nos permiten tener una plataforma totalmente configurable, pudiendo añadir nuevos algoritmos de captura en un futuro sin apenas esfuerzo o modificar cualquier elemento hardware de la plataforma, como pueden ser los puertos de lectura, y notificarlo al programa a través del script correspondiente, haciendo a la plataforma totalmente maleable y versátil.

La sintaxis que se utiliza en este script para la configuración de la plataforma es exactamente la misma que la utilizada en los scripts de configuración de los experimentos:

- palabra clave:valor asignado

La palabra clave hace referencia a una característica configurable de la plataforma, el símbolo ":" actúa como un separador para que el análisis del script sea más fácil y el valor asignado se corresponde con el valor con el que se quiere configurar una característica de la plataforma.

La modificación de los módulos software también implica la migración de los códigos de la plataforma de la versión 2 de Python a la versión 3, siendo este uno de los objetivos propuestos en (Ortega, 2017).

3.1.1. PyHuele

Como se indicó en la sección 2.4, PyHuele es el programa principal que se utiliza para la captura de los datos. Este programa ha sufrido unas ligeras modificaciones, que consisten en el uso de un segundo script para la realización de la configuración de la plataforma. El programa principal recibe este script por medio de los argumentos que se le pasan en el momento de la ejecución, y entrega este script al módulo de tratamiento de cadenas para que lo analice para finalmente configurar la plataforma como se haya especificado durante la captura de los datos.

3.1.2. Módulos del software

Como se explicó en la sección 2.4, el software de la plataforma está formado por tres módulos que permiten la captura, graficación y análisis de los ficheros de la plataforma así como el programa principal PyHuele.

3.1.2.1. Módulo de captura de datos

Este módulo es el encargado de captar y procesar los datos del sensor, para después guardarlos en ficheros. Este módulo es en el que más cambios estructurales se han realizado, además de para añadir el nuevo algoritmo de adquisición de los datos, para sanear el código y reducir el número de ficheros que lo componen lo máximo posible.

Aprovechando la incorporación del nuevo algoritmo de adquisición de datos, se ha utilizado la orientación a clases y la herencia del lenguaje de programación Python para reestructurar todo el código y hacerlo más entendible.

Para realizar la reestructuración de este módulo se ha utilizado el patrón de diseño software strategy (Phillips, 2010). Este patrón consiste en la implementación de un problema mediante diferentes soluciones, implementando una en cada objeto, para después escoger entre las múltiples posibles, ver figura 3.2. En el ámbito de este contexto, el problema inicial consiste en la captura de los datos y las soluciones posibles a este problema son las diferentes modulaciones que se tienen implementadas.

En la clase modulaciones, se encuentra todo el código que es común a las diferentes clases, junto con las variables que también son comunes para todas las clases. En esta clase se encuentran definidas las rutinas necesarias para la captura de los datos, mientras que en las clases propias de cada modulación se implementa el método de captura exacto.

Para evitar que el programa finalice, de forma abrupta, al cerrar la terminal donde se lanzó, se ha añadido la *daemonización* automática del programa. Inicialmente, el usuario tenía que hacer uso de la rutina: *nohup*, o utilizar el símbolo '&", seguido de la introducción del identificador del proceso ejecutado para la desvinculación de proceso de la terminal.

Para la escritura de los datos en ficheros se ha utilizado un hilo de ejecución aparte, del mismo modo que el utilizado para la temperatura y humedad ambiental, evitando así el trato con el sistema de entrada y salida, ya que puede retrasar al algoritmo produciendo un fallo por no respetar las frecuencias de captura. Inicialmente, la escritura de los datos capturados se hacia en el mismo hilo de captura y se contabilizaba dentro del tiempo estipulado para la captura de una muestra, pudiendo dar fallos aleatorios debido a que no se respetaba el periodo de captura. En la figura 3.3 se muestra un esquema funcional de los hilos del módulo de captura.

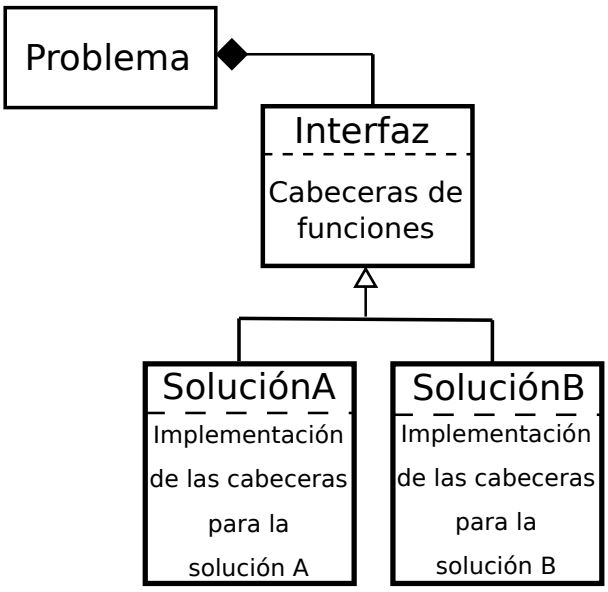


Figura 3.2: Imagen que representa el patrón strategy. En ella se tiene un problema que se busca resolver, implementando para ello varias soluciones en diferentes objetos, utilizando las mismas cabeceras que se encuentran en una interfaz o clase padre común a las diversas soluciones.

3.1.2.2. Módulo de visualización de datos

Este módulo consta solamente de un script de ejecución con todas las rutinas para la representación gráfica de los datos captados de la plataforma. Estas rutinas se han modificado debido al alto número de parámetros y a su gran complejidad de sus métodos. El resultado es que estos métodos se han reducido dando lugar a un mayor número de métodos, pero reduciendo su complejidad en gran medida.

3.1.2.3. Módulo de recogida y tratamiento de datos

Este modulo está compuesto por un solo script de ejecución. Este contiene todas las rutinas necesarias para el tratamiento de los scripts de ejecución. Este módulo ha sufrido modificaciones con el objetivo de simplificar los métodos y añadir nuevas funcionalidades, aumentando la funcionalidad de la plataforma, y permitiendo modificar valores de la plataforma que inicialmente eran estáticos.

Debido a los cambios realizados, se ha modificado, por completo, el tratamiento de ambos scripts de configuración de la plataforma. Se ha modificado la forma de dividir las expresiones de los scripts de configuración, pasando de utilizar el símbolo "," a utilizar los símbolos "{}". Una vez dividida la expresión, se genera un árbol binario para analizar que la expresión este bien formada. Cada vez que, al analizar la expresión ya dividida, se encuentre una llave de apertura o de cierre, se crea un hijo en el árbol, guardando en cada nodo un elemento. De esta forma se simplifica el proceso de análisis de los scripts con respecto al análisis inicial.

3.1.3. Pseudo-lenguaje

Para la ampliación del pseudo-lenguaje se ha realizado un análisis de aquellos elementos que intervienen en la configuración de la plataforma durante la captura de los datos y que por lo tanto pueden configurarse:

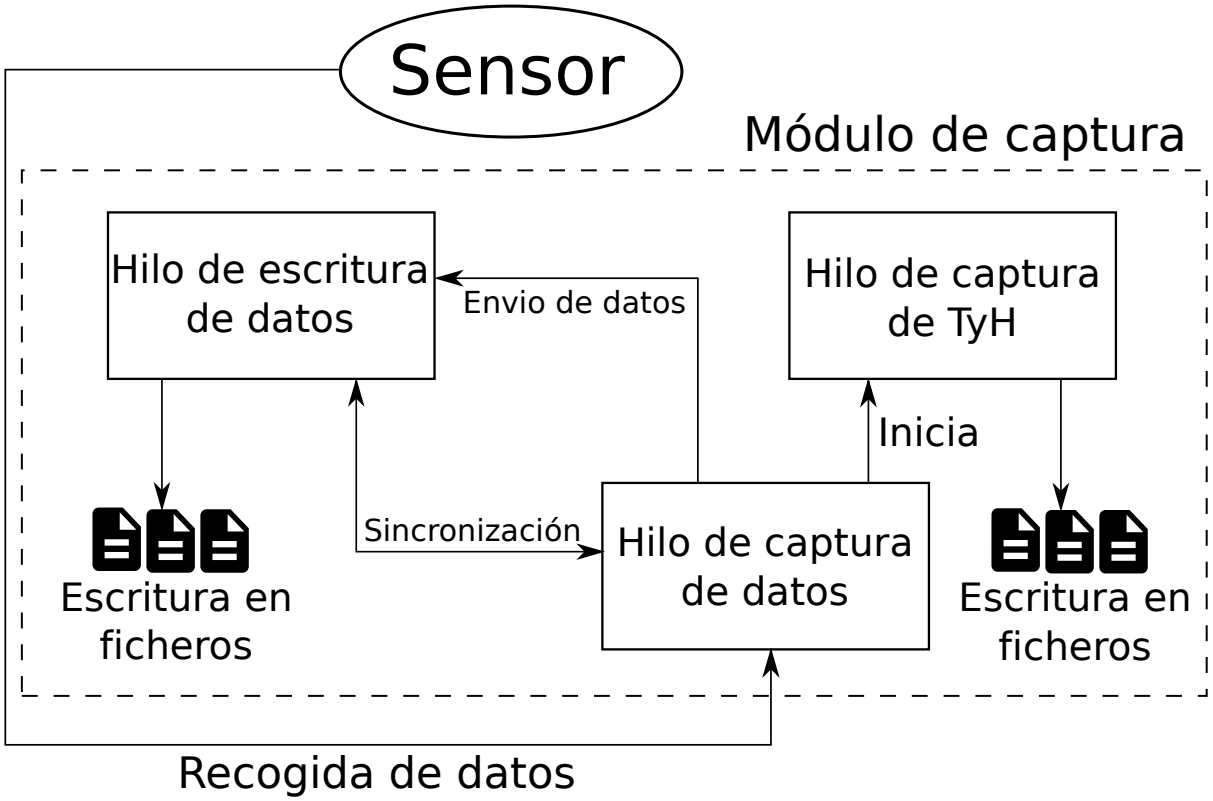


Figura 3.3: Esquema funcional de los hilos del módulo de captura. En total se utilizan tres hilos: el de captura que recoge datos del sensor e inicia el hilo de captura de TyH, el hilo de escritura de datos en ficheros, que recibe los datos del hilo de captura y los escribe en los ficheros oportunos y el hilo de captura y escritura de la TyH, que recoge y escribe los datos obtenidos en ficheros de forma independiente.

- El valor de la resistencia de carga del sensor.
- El puerto de calentamiento del sensor.
- El puerto de alimentación del motor
- El puerto de lectura del sensor.
- Carpeta donde guardar los datos recogidos del sensor.
- Posición de las válvulas.
- Número de sub-muestras que confortan un valor.
- Tiempo límite de captura de un valor.
- Valor de voltaje que se aplica al sensor de captura.
- El puerto de lectura del sensor de temperatura y humedad ambientales.
- El modelo de sensor de temperatura y humedad ambientales.
- Los puertos de activación de las electroválvulas.

Estas características configurables tienen asignadas unos valores por defecto, de tal forma que si al lanzar un experimento no se configuran todas las opciones, se utilicen los valores

por defecto, y pueda realizarse la ejecución. Además de las opciones configurables anteriores, también se ha ampliado el pseudo-lenguaje al añadir la nueva modulación basada en el uso del PID (Herrero-Carrón et al., 2015). Al igual que para las modulaciones anteriores, todas las opciones configurables tienen definido el valor, por defecto, de 0. Las características que intervienen en este modo de adquisición de datos son:

- El periodo de la señal de referencia.
- El límite máximo superior de la señal de referencia.
- El límite mínimo superior de la señal de referencia.
- El límite máximo inferior de la señal de referencia.
- El límite mínimo inferior de la señal de referencia.
- El valor de alfa.
- El valor de pico máximo de la señal de referencia.
- El valor de pico mínimo de la señal de referencia.
- El valor inicial de la temperatura del sensor.

3.1.3.1. Nuevas palabras claves del pseudo-lenguaje

Para plasmar todas las características anteriores se han definido las siguientes palabras claves.

3.1.3.1.1 Modulación PID

- period: define, en segundos, el periodo de la onda sinusoidal que se utiliza como referencia para la captura de datos.
- upper_limit_max: define el valor máximo superior de la señal de referencia.
- upper_limit_min: define el valor mínimo superior de la señal de referencia.
- lower_limit_max: define el valor máximo inferior de la señal de referencia.
- lower_limit_min: define el valor mínimo inferior de la señal de referencia.
- alpha: define el valor de la constante alfa. Este valor es utilizado para fijar el valor de la constante proporcional.
- max_peak_value: define el valor máximo de la señal de referencia.
- min_peak_value: define el valor mínimo de la señal de referencia.

3.1.3.1.2 Configuración de la plataforma

- reading_port: indicamos el puerto de lectura que se va a utilizar para leer los datos del sensor. Este puerto debe ser un puerto ADC para poder digitalizar la señal analógica que recibimos del sensor. Según la técnica de modulación escogida, esta opción puede tener los siguientes valores: *P9_38*, *P9_40* o *P9_12*.

- heating_port: definimos el puerto que se va a utilizar para calentar el sensor de captura de datos. Este puerto debe ser un puerto PWM para poder variar el rango de voltaje que recibe el sensor. El puerto por defecto utilizado es el *P9_22* o el *P9_14*, según el tipo de modulación escogida.

- motor_pin: definimos el puerto de alimentación del motor de succión. Este puerto debe ser un puerto PWM para poder variar el rango voltaje con que alimentamos el sensor. El puerto por defecto utilizado es el *P9_21*.

- resistance: indicamos el valor de la resistencia de carga que se utiliza en el sensor de captura de los datos. El valor de la resistencia utilizado por defecto es: *440* o *27000*.

- sd_folder: representa la ruta donde van a guardase los datos obtenidos del sensor. Recibe este nombre por convenio, ya que por seguridad los datos se guardan en una carpeta dentro de una tarjeta SD para evitar pérdidas de datos. El nombre por defecto utilizado para nombrar las carpetas, donde se guardan las muestras, depende de la modulación. El esquema utilizado es: *CAPTURAS/NOMBRE_MODULACION*.

- valve_position: indica el modo de apertura de las válvulas. Debido a la adquisición de un nuevo conjunto de válvulas, para la realización de un TFM paralelo a este, se ha añadido esta opción ya que la base de este nuevo conjunto de válvulas admite el uso de varias posiciones. El valor que recibe es una cadena con el valor "P" o "R" indicando las posiciones de apertura de las válvulas. La posición de las electro-válvulas por defecto es: *P*.

- number_samples: define el número de sub-muestras que componen una muestra. Para tomar una muestra, se capturan un conjunto de sub-muestras en un tiempo acotado, se promedian y el resultado es la muestra que se guarda. El valor por defecto es: *10*.

- sleep_time: define la frecuencia de captura de las muestras del sensor. Indica el tiempo, en segundos, que pasa entre la realización de la captura de dos valores del sensor. El valor por defecto es un segundo.

- sleep_time_th: define la frecuencia de captura de las muestras del sensor de temperatura y humedad. Indica el tiempo, en segundos, que pasa entre la captura de dos valores del sensor. Su valor por defecto es 59 segundos.

- vcc: define el valor del voltaje que se aplica al sensor de captura de datos. El valor asignado por defecto es: *5*.

- th_reading_port: define el puerto de lectura de los datos del sensor de temperatura y humedad. El puerto que se utiliza por defecto es el: *P8_11*.

- electrovalves_port: define los puertos con los que se van a controlar la apertura y el cierre de las electroválvulas. Estos puertos se defienen mediante una lista de puertos separados por comas. Todos los puertos deben ser del tipo GPIO. Los puertos definidos por defecto, para las electro-válvulas son: *P8_10*, *P8_12*, *P8_14* y *P8_16*.

- type_sensor: indica el modelo del sensor de temperatura y humedad ambiental que se utiliza en la plataforma. El modelo de sensor indicado, por defecto, es: *AM2302*.
- reading_time: define la frecuencia de captura, en segundos, de las sub-muestras que componen una muestra. La frecuencia de captura utilizada por defecto es: *0.1*.

3.1.3.2. Sintaxis del pseudo-lenguaje

La sintaxis del pseudo-lenguaje también se ha modificado, con el objetivo de añadir un conjunto de directrices nuevas y facilitar al usuario la creación de los scripts de ejecución. Para ello se ha modificado una parte de la simbología original del pseudo-lenguaje. Este cambio afecta a la forma en que se abren varias válvulas simultáneamente. Inicialmente, para abrir varias válvulas al mismo tiempo, se indica mediante el símbolo ".". Se ha modificado la simbología por el símbolo "+".

- Para realizar la apertura de varias válvulas inicialmente se utilizaba la notación: 1.2.4. Esto abría las válvulas 1,2 y 4 al mismo tiempo.
- Para realizar la apertura de varias válvulas actualmente se utiliza la notación: 1+2+4. Esto abre las válvulas 1,2 y 4 al mismo tiempo.

Debido a este cambio, se ha introducido el uso de número reales en el pseudo-lenguaje, aumentando el rango de valores que es posible utilizar en los scripts de ejecución.

Por último, se ha añadido una directriz para la creación de secuencias ordenadas de valores numéricos sin necesidad de tener que indicar todos los valores de la lista. La sintaxis es la siguiente:

- inicio;fin;salto

Cada elemento se separa del siguiente mediante el uso del símbolo ";". Uno ejemplo de uso es el siguiente:

- 1;10;2

Esto es lo mismo que escribir en el script:

- 1,4,7,10

El valor de salto define la cantidad de valores que avanzamos para obtener el siguiente valor de la secuencia. Este parámetro no es obligatorio utilizarlo, y si no se indica el valor que se utiliza es la unidad. Los valores de inicio y de fin de secuencia se encuentran ambos en la secuencia retornada.

3.2. Interfaz gráfica

La interfaz gráfica permite la iteración con la plataforma de captación de odorantes de forma visual, sin necesidad de tener que usar la linea de comandos, ni de conectarse a través de esta a la plataforma. De esta forma un usuario, no experto en el funcionamiento interno de la plataforma, puede realizar experimentos fácilmente. Esta interfaz supone una capa de abstracción sobre el programa principal PyHuele, ya que cuando se inicie el proceso de captura,

la interfaz se conectará de forma remota a la plataforma y llamará a PyHuele, pasándole los datos requeridos para realizar la captura de los datos.

Este interfaz se ha diseñado siguiendo el esquema MVC (Modelo-Vista-Controlador), ver figura 3.4. En este patrón de arquitectura software, se separan la lógica del programa de los datos y de la representación gráfica, permitiendo generar un código reutilizable y limpio para futuros cambios. En este patrón de diseño, el modelo es el encargado de gestionar los datos y de realizar la lógica. El controlador se encarga de responder a los eventos que genere el usuario invocando al modelo o modificando la vista. Por último, la vista representa el modelo de forma gráfica para la iteración con el usuario. La figura 3.5 muestra un esquema de la interfaz.

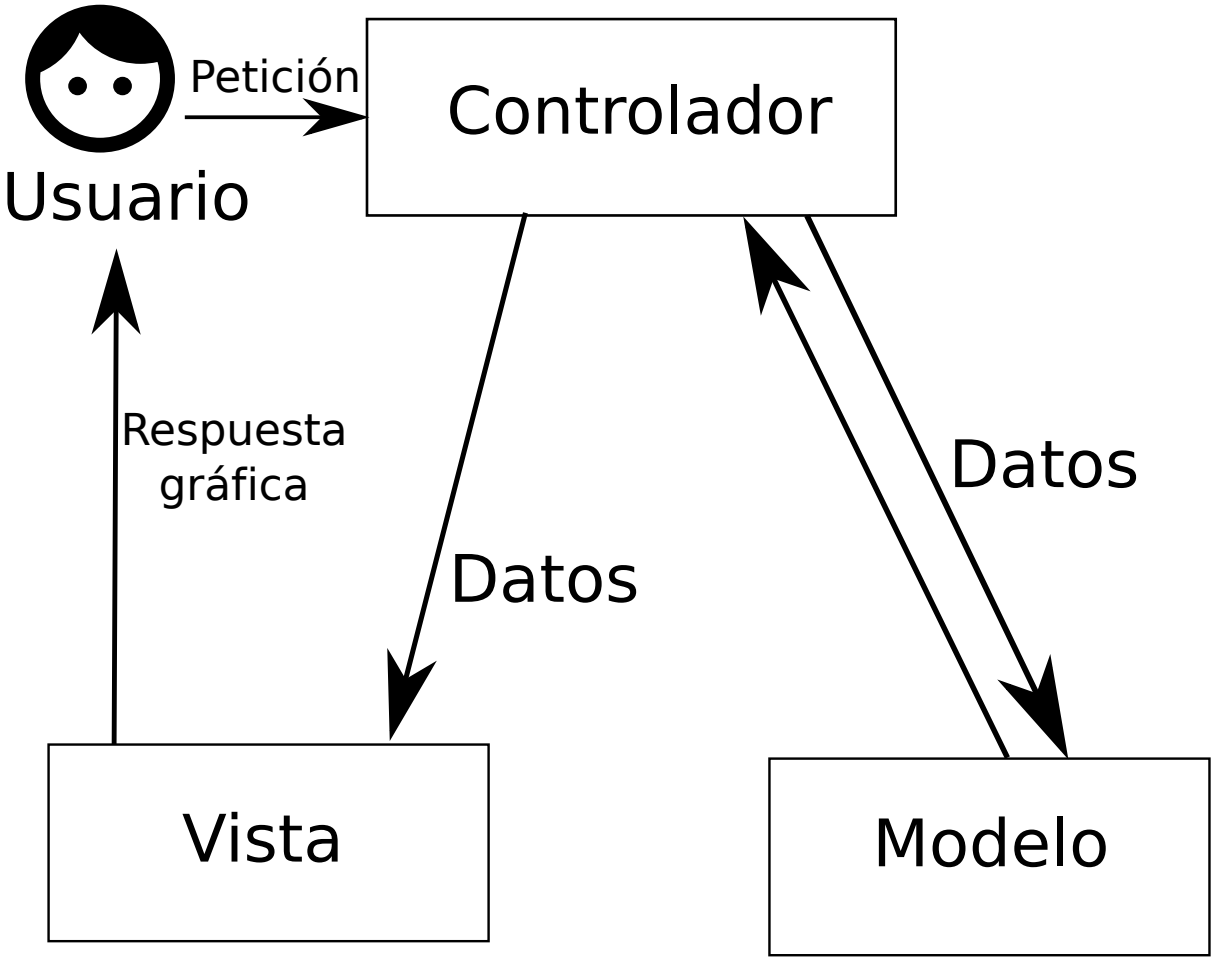


Figura 3.4: Imagen que representa el patrón de diseño Modelo-Vista-Controlador.

Para implementar este modelo, se ha separado cada elemento que conforma la interfaz en una clase diferente, respetando el esquema MVC, explicado en la sección 3.2. Los requisitos fundamentales que debe cumplir esta interfaz son los siguientes:

- RF 1. La interfaz debe permitir el uso del pseudo-lenguaje que utiliza la plataforma.
- RF 2. La interfaz debe permitir cargar el contenido de un script de ejecución o de configuración.
- RF 3. La interfaz debe permitir guardar las opciones introducidas en un script
- RF 4. La interfaz debe de conectarse a cualquier plataforma y lanzar los experimentos requeridos.

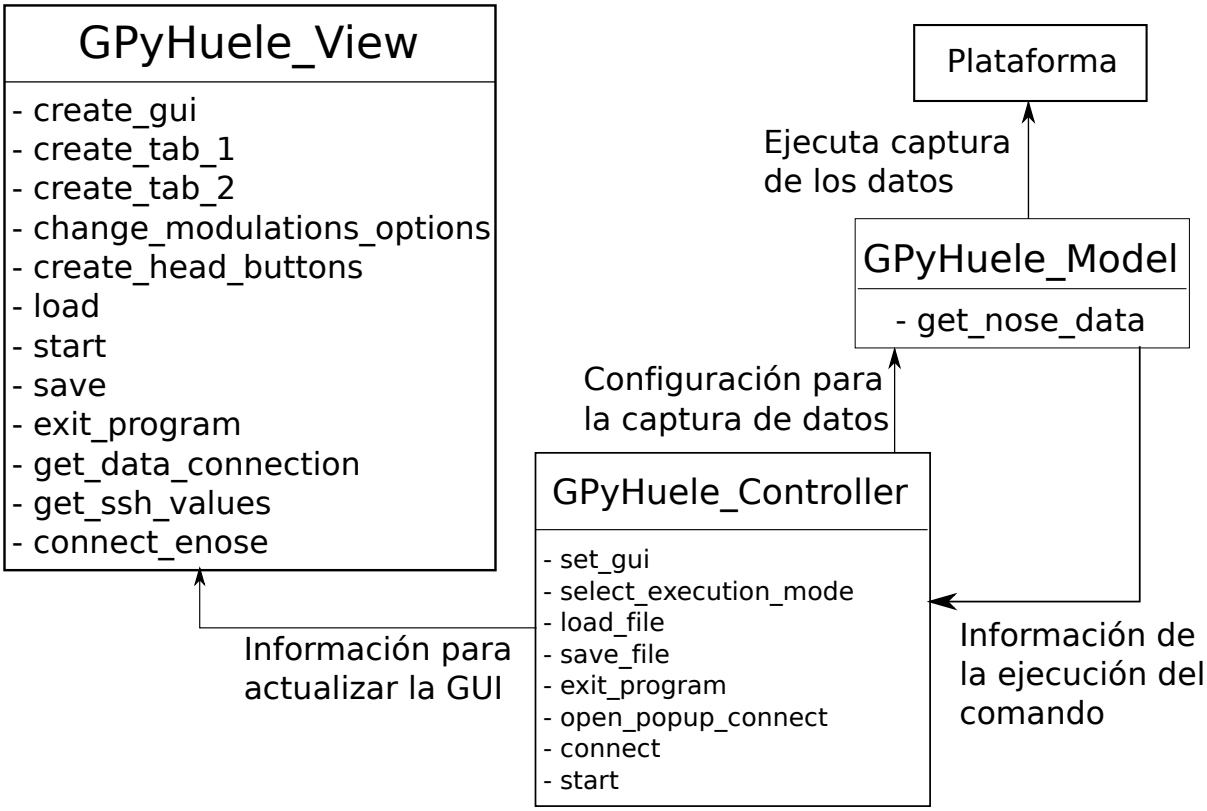


Figura 3.5: Esquema de la interfaz gráfica desarrollada.

- RNF 1. La interfaz debe ser user-friendly, permitiendo que el usuario pueda utilizarla intuitivamente.

- RNF 2. El diseño se hará modular para un fácil entendimiento del código, y en caso de ampliación, se realice de forma simple.

El diseño de la interfaz tiene que ser simple y albergar todas las opciones para poder configurar tanto los experimentos que se lancen, como la plataforma durante la captura. Para poder albergar todas las opciones disponibles y no sobrecargar la vista, se ha utilizado dos pestañas, una para la configuración del experimento, y otra para la configuración de la plataforma. En la pestaña de configuración de los experimentos habrá un desplegable que permita la selección de los diferentes tipos de modulación, actualizando la vista y añadiendo las directrices de configuración propias para cada tipo de modulación. En ambas pestañas se encuentran las etiquetas y las entradas para las diferentes opciones que puedan introducirse.

Según en la pestaña en la que nos encontremos, podemos realizar la carga o el guardado de los datos de configuración del experimento o de la plataforma, de tal forma que si nos encontramos en la pestaña de configuración del experimento, podemos cargar ficheros o guardar solo los datos que configuran el experimento, y si estamos en la pestaña de configuración de la plataforma solo se podrá cargar o guardar ficheros que configuren la plataforma.

También contará con un conjunto de botones en la parte superior que permitirá cargar o guardar datos, definir los parámetros de la conexión remota a la plataforma, lanzar el experimento de forma remota o salir.

3.3. Modelización de las neuronas del bulbo olfativo para la extracción de características

Para la modelizar las neuronas de la red biológica, explicada en la sección 2.7.2, implicadas en la extracción de las características de las curvas de odorantes capturadas, se ha utilizado el modelo neuronal propuesto por Izhikevich (Izhikevich, 2003), que es el mismo que se utiliza en (Jing et al., 2016). Se ha utilizado este modelo neuronal porque no es muy costoso computacionalmente, ya que solo consta de dos ecuaciones, evitando así largas esperas de tiempo por la ejecución de este modelo. Las ecuaciones del modelo que simulan el potencial de membrana son las siguientes:

- $$C\dot{v} = k(v - v_r)(v - v_t) - u + I(t).$$
- $$\dot{u} = a[b(v - v_r) - u].$$

También tiene dos ecuaciones que modelan la recuperación de la membrana, que se utilizan cuando el valor del potencial de membrana ha pasado el umbral de disparo:

- $$v = c.$$
- $$u = u + d.$$

El significado de las diferentes incógnitas es el siguiente:

- v: representa el potencial de membrana.
- u: es la variable que modela la recuperación de la membrana.
- C: simula la capacitancia de la membrana.
- v_r: es el potencial de reposo de la membrana.
- v_t: simula el potencial de umbral instantáneo.
- I(t): imita la inyección de potencial externo a la neurona.
- a: describe la escala de tiempo de la variable de recuperación u.
- b: describe la sensibilidad de la variable de recuperación, u , a las posibles fluctuaciones del potencial de membrana.
- c: representa el valor de reinicio del potencial de membrana tras un spike.
- d: variable de reinicio tras un spike.
- v_{peak}: valor al que se produce un spike, reiniciando el potencial de membrana.

Parámetros	ORN	MC	PC	GC	SAC
C	25	40	5.9	7.1	58
k	1	1	0.049	0.058	0.061
v_r	-55	-55	-53.1	-50	-67
v_t	-50	-50	-20	-20	-30
a	0.4	0.4	0.0167	0.0167	0.049
b	2.6	2.6	-0.94	-0.94	-0.68
c	-50	-50	-20	-20	-30
d	200	200	50	50	150
v_{peak}	35	35	35	35	35

Tabla 3.1: Parámetros utilizados en el modelo del bulbo olfativo obtenidos de (Jing et al., 2016). Para obtener un resultado óptimo, en trabajos futuros, se debe realizar una búsqueda de los valores para los cuales se consigue un desacople óptimo de las señales neuronales generadas.

En este modelo, las ecuaciones que simulan el potencial de membrana y la variable de recuperación son ecuaciones diferenciales, ya que dependen de un término independiente, que es el tiempo. Como solo dependen de una variable independiente, son ecuaciones diferenciales ordinarias (EDO), y para poder calcular estas series se necesita utilizar un integrador y un paso de integración.

Como valores de los diferentes parámetros de las ecuaciones se han utilizado los propuestos en el trabajo (Jing et al., 2016). Estos se muestran en la tabla 3.1 y se han obtenido de los trabajos (Polese et al., 2014; Izhikevich, 2003, 2007), aunque pueden ser modificados para tratar de obtener mejores resultados. Como valores iniciales para las diferentes neuronas se han utilizado los propuestos en (Izhikevich, 2003). En un futuro se buscará obtener los parámetros con los cuales las señales obtenidas estén completamente desacopladas, teniendo en cuenta que la plataforma solamente dispone de un solo sensor, mientras que en el artículo (Jing et al., 2016) utilizan un total de 10 sensores.

4

Desarrollo e integración del entorno y las técnicas de procesamiento en NE

En este capítulo se expone todo el desarrollo software que se ha realizado: los cambios en los módulos software, la interfaz gráfica y la programación de las rutinas, para la extracción de características, utilizando el modelo de red neuronal, explicado en la sección 2.5.4. Todo el código relacionado con la plataforma de captación de odorantes puede encontrarse en el github: `https://github.com/TheBarto/EN/tree/master`.

4.1. Tecnologías utilizadas

La plataforma utilizada esta compuesta por una BBB (Instrument, 2013), con un sistema operativo Debian 7.11. Este SO se encuentra adaptado para su uso en este tipo de sistemas embebidos, permitiendo el uso de los múltiples pines de entrada y salida de los que dispone la BBB, y excluyendo varios programas que vienen por defecto en esta distribución. La versión de Python necesaria para el uso del software, Python 3.3 o superior, no viene instalada por defecto, y hay que instalarla para la ejecución de los diversos scripts de captura, así como ciertas librerías complementarias. La instalación y puesta a punto de la plataforma se muestra en el anexo A.

4.2. Software de experimentación

El software original de experimentación ha sufrido variaciones, como se comentó en la sección 3.1. Para llevar a cabo estos cambios se ha utilizado la orientación a clases y la herencia de Python, además de múltiples estructuras ya implementadas en las librerías instaladas.

4.2.1. Módulo de recogida de datos

En este módulo se ha modificado sobre todo la estructura organizativa, pasando de tener cuatro scripts, tres para las diferentes modulaciones y uno para el código común, a unificarlo todo en un script llamado: *modulaciones.py*, que puede encontrarse en el anexo C. Para ello se

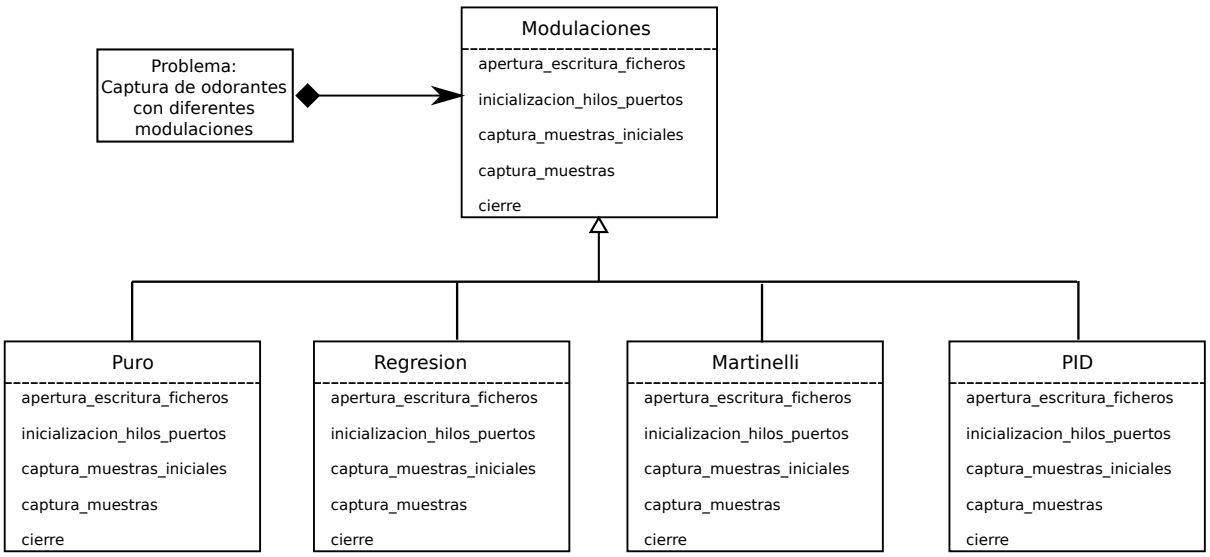


Figura 4.1: Aplicación de la orientación a clases y del patrón strategy al módulo de captura de datos.

utilizó el patrón strategy (Phillips, 2010), pasando a tener un script con 5 clases, una para cada tipo de modulación y la clase padre, como se muestra en la figura 4.1.

En la clase padre, que se ha llamado *Modulaciones*, se encuentran todos los métodos y variables que definen los parámetros de captura y configuración de la plataforma, comunes a las diferentes modulaciones.

El código común lo conforman todos los métodos de apertura y cierre de ficheros y electroválvulas, la iniciación de los pines de entrada y salida de la plataforma, el cierre de los hilos utilizados y la captura de los datos de la temperatura y humedad ambiental. En la clase padre se incluyen las cabeceras de aquellos métodos que son necesarios para realizar la captura de los datos, y que por lo tanto deben de sobre-escribirse para añadir un nuevo modelo de modulación:

- **apertura_escritura_ficheros**: se encarga de crear los ficheros y abrirlos para su escritura posterior.
- **inicializacion_hilos_puertos**: se carga la configuración de los puertos, definiendo si son de entrada o salida y se ejecutan los hilos de escritura y de lectura de la temperatura y humedad ambiental.
- **captura_muestras_iniciales**: se captura un conjunto inicial de muestras que será utilizado después por los algoritmos.
- **captura_muestras**: se capturan las muestras de los diferentes odorantes.
- **cierre**: cierra todos los ficheros e hilos abiertos y reinicia todas las variables utilizadas para la siguiente captura.

Todos estos métodos son llamados desde la función: *captura_datos*, que se encuentra en la clase padre *Modulaciones*, siendo este el método que se encarga de realizar el proceso de captura.

Estos métodos cuentan con algunas líneas de código comunes a todas las clases, evitando así su repetición en las diferentes implementaciones realizadas. Estas funciones se han reimplementado en las clases de las modulaciones, adaptándolas a los diferentes algoritmos de captura. Además de estos métodos, cada algoritmo de modulación tiene sus propios métodos para realizar

la captura de los datos. Ocurre lo mismo con las variables de configuración de los experimentos, ya que hay ciertos parámetros que definen la captura de los datos que son propios de un tipo de modulación, y que por lo tanto las variables en las que se almacenan estos parámetros se definen en las clases de las modulaciones correspondientes.

4.2.1.1. Comunicación entre hilos

Para realizar la comunicación y envío de datos del hilo de captura con el hilo de escritura se ha utilizado la clase de Python Queue[1]. Esto se ha hecho para no entorpecer la captura de los datos. Esta clase es muy útil en el uso de programación con hilos, ya que permite el intercambio de información entre hilos de forma fácil y segura, por lo que es perfecto para su uso en este caso. Cada vez que el hilo de captura recoge un nuevo valor de voltaje, lo guarda en una lista junto con la resistencia y la temperatura del sensor, y lo envía por la cola al hilo de escritura. El hilo de escritura recibe esta lista, prepara las cadenas correspondientes y escribe los datos en los ficheros pertinentes. Si al realizar la lectura de la cola, esta está vacía, el hilo de lectura se queda bloqueado hasta que se vuelva a introducir un elemento en la cola. Para sincronizar los hilos se ha utilizado el sub-módulo Event[2] de la clase Threading de Python. Esta clase tiene una bandera propia que se fija a True o False. Mediante la llamada a la función bloqueante, según el estado de la bandera el hilo se quedará bloqueado o continuará su ejecución, de la misma forma que un semáforo, pero con una interfaz más simple. Se ha seleccionado este método cuando se deban cerrar los ficheros donde se almacena la información recogida, y haya que sincronizar los hilos de captura y de lectura. De este modo, se bloquea el hilo de captura, permitiendo al hilo de escritura almacenar todos los datos que tenia en la cola de mensajes. Una vez que se han escrito todos los datos, el hilo de captura se desbloqueará y el proceso de captura y escritura, de los datos, vuelve a empezar.

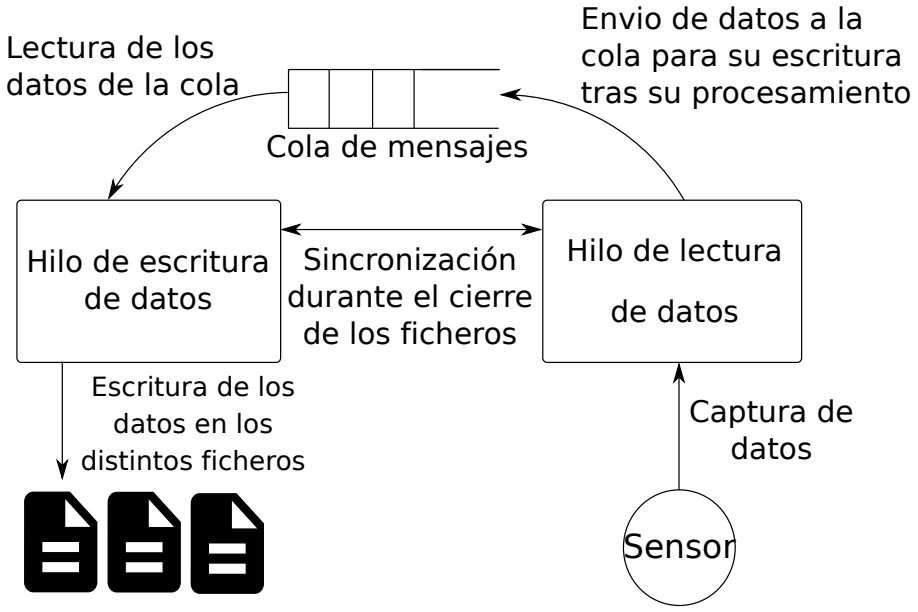


Figura 4.2: Esquema del funcionamiento de los hilos de captura y escritura de los datos capturas del sensor.

[1]https://docs.python.org/3/library/asyncio-queue.html
[2]https://docs.python.org/3/library/threading.html#threading.Event

4.2.1.2. *Daemonización* del programa principal

Para realizar la *daemonización* del programa, lo primero que se ha hecho ha sido realizar una llamada a la función fork, creando así un proceso hijo igual al proceso padre. Tras haber creado este proceso hijo, se comprueba el pid de ambos procesos, y se finaliza el proceso cuyo pid es mayor que 0, que se corresponde al pid del padre. En el proceso hijo, se cambia la máscara de permisos poniéndola a 0, con la función os.umask(0), dejando que el proceso hijo pueda crear, leer y escribir en ficheros libremente, ya que con una máscara heredada los permisos pueden ser más restrictivos. Después le convertimos en líder de la nueva sesión, así como líder del grupo de un nuevo grupo de procesos y se le desvincula de la terminal de control, mediante la instrucción: os.setsid(). Después de realizar este proceso, finalizamos realizando un segundo fork, por precaución, ya que el proceso puede volver a vincularse a una terminal al abrir una sin la opción O_NOCTTY, impidiendo que el proceso se vincule con una terminal. Por último, se vacían los buffers de las salidas y entradas, sys.stdout, sys.stderr y sys.stdin, del proceso y se redirigen a unos ficheros. Todo este proceso se muestra en el código 4.1 (Gift and Jones, 2008).

```
def daemonizar(stdin='stdin.txt',stdout='stdout.txt',
    stderr='stderr.txt'):
    try:
        pid = os.fork()
        if pid > 0:
                sys.exit(0)
    except OSError as e:
        sys.sderr.write("fork #1 failed: (%d) %s\n"
            % (e.eerno, e.streerror))
        sys.exit(1)

    os.umask(0)
    os.setsid()
    try:
        pid = os.fork()
        if pid > 0:
                    sys.exit(0)
    except OSError as e:
        sys.sderr.write("fork #2 failed: (%d) %s\n"
            % (e.eerno, e.streerror))
        sys.exit(1)

    for f in sys.stdout, sys.stderr,sys.stdin: f.flush()
    si = open("salida_in.txt", 'r')
    so = open("salida_out.txt", 'w')
    se = open("salida_err.txt", 'w')
    os.dup2(si.fileno(), sys.stdin.fileno())
    os.dup2(so.fileno(), sys.stdout.fileno())
    os.dup2(se.fileno(), sys.stderr.fileno())

    pid = str(os.getpid())
    open('file_daemon','w+').write("%s\n" % pid)
```

Listing 4.1: Código que muestra el proceso seguido para la *daemonización* del proceso de captura de datos.

Para pasar las características de configuración de la plataforma se ha utilizado, de la misma

forma que para el paso de las características del experimento, la estructura diccionario que tiene implementada Python. Esta estructura de Python permite almacenar pares clave-valor pudiendo pasar al software de la plataforma que características de esta queremos configurar y el valor que les queremos asignar de una sola vez, siendo la opción más cómoda para realizar el paso de las características. Una vez que el software ha recibido el diccionario, se le pasa a la clase *Modulaciones* y esta se encarga de ir leyendo todos los pares clave-valor y guarda los diferentes valores en variables comunes a todas las clases. En el caso de no haber definido alguna característica, se utilizará un valor por defecto, mostradas en el capitulo 3.

4.2.2. Modulo de recogida y tratamiento de datos

Como se explicó en la sección 3.1.2.3, los cambios que se han realizado a este módulo se han orientado a la adición de las nuevas características del pseudo-lenguaje, junto a la configuración de la plataforma, a la hora de realizar experimentos. Aprovechando la introducción de nuevos elementos en la sintaxis del pseudo-lenguaje, los cuales se mostraron en la sección 3.1.3.2, se ha cambiado por completo las rutinas de tratamiento de los scripts.

Lo primero que se ha modificado fue la localización de las variables, que contienen las palabras claves en el código, ya que inicialmente se encontraban dentro de un conjunto de clausulas if/else gigantesco, pasando a guardarlas en listas fuera de las funciones, haciendo más escalable el código. En total se tienen cinco listas: una para las palabras clave que tienen en común todos los algoritmos de modulación, otra para las palabras claves que sirven para configurar la plataforma y cada una de las restantes albergan las palabras claves propias para cada modulación. Cuando se está procesando un script que contiene las características del experimento, se utiliza la lista que contiene las palabras claves comunes a las diferentes modulaciones, y del diccionario de configuración del experimento, obtenemos el tipo de modulación deseado y usamos la lista con las palabras clave pertinentes.

Para almacenar, temporalmente, las características del experimento y de la plataforma, una vez leídos ambos scripts de configuración, se utiliza la estructura diccionario de Python. Mediante el uso de esta estructura se puede almacenar pares clave-valor, pudiendo pasar al software todas las características de configuración fácilmente. Cuando el software recibe ambos diccionarios, estos se entregan a la clase padre, que va leyendo y guardando la información que contiene. En caso de no haber definido una característica, al guardar los datos en el diccionario, se le otorga un valor por defecto. Un problema que tienen los diccionarios es su lentitud de acceso a los datos que contienen, debido al uso de funciones hash, que mapean la clave utilizada a un valor numérico. Por este motivo, se ha decidido acortar las palabras utilizadas como clave, reduciendo así el tiempo de acceso a los datos.

Debido al procesamiento de otro script, se ha codificado un nuevo conjunto de funciones, ya que aunque el procesamiento es casi el mismo, hay varias diferencias entre el procesamiento de ambos scripts. Esto ha dado lugar a dos conjuntos de funciones, tanto para la lectura de los scripts:

- leer_fichero_experimento.
- leer_fichero_conf_plataforma.

como para el procesamiento del contenido de los scripts:

- tratamiento_datos_experimento.
- tratamiento_datos_plataforma.

Las funciones de lectura engloban a la función: *leer_fichero*, que simplemente lee el fichero, para después procesar el contenido del fichero. Todas las funciones se llaman desde la función: *tratamiento_datos*, que engloba todo el proceso de lectura y tratamiento de los scripts.

El análisis de los scripts se ha modificado por completo, como se indicó en la sección 3.1.2.3, pasando a utilizar el TAD árbol binario. Este TAD simplifica el análisis de las expresiones, que configuran tanto el experimento como la plataforma, y permiten comprobar si están bien formadas. Para implementar este TAD se han utilizad las listas de Python. La estructura principal la compone una lista genérica, y los nodos del árbol están compuestos por listas, donde se almacenan la posición del nodo padre en la lista del árbol, la expresión asignada al nodo y las posiciones asignadas a los hijos izquierdo y derecho.

Para la creación del árbol, lo primero es dividir la expresión por los símbolos "{}". Si al analizar las sub-expresiones, se encuentra en una el símbolo "{", se baja por la rama izquierda del nodo actual. Si, por el contrario, se encuentra el símbolo "}", se sube al nodo padre, y se baja por su rama derecha. En cada nodo se almacena la sub-expresión analizada en ese momento.

Para analizar el árbol generado a partir de la expresión, se comienza por el nodo raíz, analizando su valor, para después descender por el nodo izquierdo, hasta que nos encontremos un nodo hoja, es decir, un nodo sin hijos. Cuando esto ocurre se sube al nodo padre y se desciende por el nodo derecho de este, para volver a repetir este proceso y analizar así todo el árbol. Las funciones que se han codificado para ello son las siguientes:

- **evaluar_expresion**: crea el árbol binario y lo evalúa. Este método es invocado al realizar el análisis del script de configuración del experimento.
- **crear_arbol**: crea el árbol binario.
- **tratamiento_expresion**: esta función recibe una sentencia del fichero de ejecución y divide su contenido por el símbolo ",". Después se encarga de buscar los símbolos "+" y "-", que son los que forman listas de elementos, como las listas de apertura de electroválvulas o el tiempo de apertura de una secuencia de electroválvulas.
- **tratar_elemento**: esta función recibe una sentencia formada por los símbolos: "x*","x:y:z","x(y)" o "x", donde: x, y, z son números reales, y analiza estas expresiones devolviendo el resultado.

4.2.3. Modulo de representación gráfica

En este módulo se ha modificado la lógica de las funciones, con el objetivo de simplificarlas lo máximo posible, dividiendo su contenido en varias funciones más simples. También se han añadido varias funcionalidades nuevas, permitiendo la representación de los datos de nuevas maneras.

Como primer paso se ha cambiado la forma original de paso de argumentos a estas funciones, ya que originalmente se utilizaba un diccionario de Python. En este diccionario se introducían todos los datos necesarios para la representación gráfica de los datos. Este diccionario se ha sustituido por el número mínimo de argumentos, necesarios para que las funciones realicen su objetivo. Para cada experimento, se puede indicar que serie representar indicando su columna en el fichero donde se almacena. Del conjunto de datos que obtenemos del sensor, podemos representar tres series diferentes:

- Voltaje
- Resistencia

- Temperatura

Debido a la posibilidad de poder representar estas tres series, se han organizado las funciones en dos grupos:

- Funciones que imprimen todas las series en la misma gráfica.
- Funciones que imprimen cada serie en una gráfica diferente.

Cada uno de estos grupos puede representar los datos de la siguiente manera:

- Imprimir toda la serie en una misma gráfica.
- Dividir la serie por las transiciones de los odorantes captados y representarlos en la misma gráfica, pero en diferentes ventanas, o subplots.

Si se escoge representar cada experimento en una gráfica diferente, y las series de cada experimento por separado, se generarán tantas gráficas como experimentos y series se hayan escogido. Si se elige representar las series, del mismo tipo, en una misma gráfica, se crearán tantas figuras como series se hayan elegido. Por último, se pueden representar, para diferentes experimentos, todas las series en una misma figura.

Como nuevas funcionalidades, se han codificado un conjunto de funciones que permite seleccionar, para una o varias carpetas, un conjunto de ficheros y representarlo de las formas anteriormente expuestas. Para poder diferenciar las diferentes carpetas seleccionadas, se ha utilizado un color diferente para cada una, y para diferenciar los experimentos de una carpeta, se han utilizado diferentes tonalidades. En caso de representar solamente una carpeta, cada experimento se representa de un color diferente.

Para poder diferenciar las diferentes carpetas y ficheros seleccionados para su representación, se han utilizado diferentes colores para las diferentes carpetas, o para los diferentes ficheros si solo se ha elegido una carpeta. Para diferenciar los diferentes ficheros pertenecientes a una misma carpeta, si se han elegido varias carpetas, se han utilizado diferentes tonalidades. Para la realización de todas estas funciones se ha utilizado el módulo pyplot de la librería de Python Matplotlib[3].

Como funciones básicas, se han desarrollado tres, utilizando el código común a las diferentes funciones originales de este módulo. Estas tres funciones contienen todo el código que permite representar y guardar los datos, de tal forma que el resto de funciones sólo tengan que llamarlas con los argumentos básicos, simplificándolas así lo máximo posible. Las funciones son las siguientes:

- **plot_lst**: esta función recibe una lista que contiene las tres series de datos que se pueden representar, junto con cual de estas series se quiere representar y las grafica.
- **treat_data**: recibe un fichero junto con una función de normalización que se aplica a los datos tras haberlos leídos, si se desea, para después representarlos.
- **save_plot**: guarda en ficheros, en la carpeta donde se le especifique, las diferentes representaciones gráficas, imprimiendo el título y los ejes que se le indiquen.

Sobre estas tres funciones se han desarrollado el resto de las funciones de la librería, tratando que estas sean lo más simples posibles, generando un conjunto de funciones que permitan la representación de los datos de diferentes maneras. El formato de las cabeceras de estas funciones es el siguiente:

[3] https://matplotlib.org/

- plot_A_B_C_D(*arg1*, *arg2*, ..., *argN*)

El significado de cada una de las incógnitas es el siguiente:

- A. file, files, folder_file, folders_files, folder_chunk o folders_chunk: indican si se quiere representar un fichero, un conjunto de ficheros, pertenecientes a misma carpeta o de diferentes carpetas, un conjunto de ficheros de una misma carpeta, indicando su posición en la carpeta o un conjunto de ficheros de diferentes carpetas.
- B. together o separately: indican si los diferentes ficheros seleccionados se representan en una misma gráfica o en diferentes gráficas.
- C. columns_together o columns_separately: indican si cada una de las columnas seleccionadas para su representación se graficarán en la misma gráfica o en diferentes gráficas, una para cada serie.
- D. div_subplot: indica que las diferentes transiciones entre odorantes se representarán en diferentes ventanas dentro de una misma gráfica.

Algún ejemplo de las funciones implementadas son:

- plot_file_columns_together: para un fichero, imprime las columnas seleccionadas en la misma gráfica.
- plot_folders_files_together_columns_together: para un conjunto de ficheros, imprime las columnas seleccionadas en la misma gráfica.
- plot_folders_files_together_columns_separately_div_subplots: imprime las columnas seleccionadas en sub-graficas por las transiciones, para un conjunto de ficheros de la misma carpeta o de diferentes carpetas.

4.2.4. Interfaz gráfica

Para el desarrollo de la interfaz gráfica se ha utilizado la librería de Python Tkinter[4]. Mediante el uso del patrón de diseño MVC (Modelo-Vista-Controlador) se han implementado tres clases diferentes, haciendo también uso del módulo de tratamiento de cadenas, del software de la plataforma. Este programa se ha llamado: GPyHuele.

Esta interfaz gráfica se basa en el paradigma del cliente-servidor, en la que el supuesto cliente es la interfaz, y el servidor es la plataforma. La interfaz puede ejecutarse en cualquier entorno, siempre que se tenga instalado Python 3 y las librerías pertinentes. La interfaz envía la información necesaria para la ejecución de los experimentos y la configuración de la plataforma a esta, y cuando la plataforma ejecuta el comando, responde a la interfaz con el resultado obtenido de la ejecución del comando, no de la finalización de este.

La primera clase implementada se ha llamado GPyHuele_View, y contiene todo el código relacionado con la vista de la interfaz gráfica, como se muestran en las imágenes 4.4 y 4.5, utilizando para ello la librería Tkinter. También se han utilizado las funciones del módulo de tratamiento de cadenas para realizar el procesamiento de los diferentes valores que se introduzcan en la interfaz. Se han incluido funciones que permiten cargar y guardar ficheros en la interfaz, junto con una función que permite recoger los datos relacionados con la conexión

[4]https://docs.python.org/3/library/tk.html

ssh, sirviéndose de un pop-up que se ha desarrollado para este fin, una función que lanza el experimento y una función que cierra el programa.

A la función que se encarga de cargar datos en la interfaz, *load*, se le indica un fichero utilizando funciones de Tkinter[5], abre el fichero seleccionado y va leyendo los valores e imprimiéndolos por pantalla. Para guardar los datos introducidos en la interfaz, se repite este proceso pero a la inversa, se van leyendo los datos de la interfaz y se van guardando en el fichero indicado. La función que realiza este proceso se llama: *save*. Ambos ficheros utilizan la sintaxis que se utiliza en la plataforma.

La función que recibe los datos de la conexión ssh, *connect_enose*, lanza una ventana, ver imagen 4.3, con los elementos necesarios que hay que rellenar para realizar la conexión ssh:

- IP de la plataforma a la que nos queremos conectar.
- El usuario de la plataforma.
- La contraseña del usuario.
- El puerto de la plataforma al que conectarse.
- La ruta donde se encuentran los módulos software del programa para ejecutar el proceso de captura.

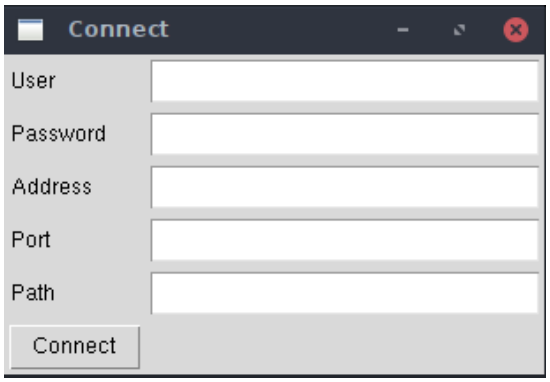


Figura 4.3: Pop-up que permite la introducción de los datos para la conexión ssh a la plataforma.

Tras recoger estos datos, GPyHuele los guarda en variables internas mediante el uso de la función *get_data_conection*.

La función que se encarga de lanzar el experimento, *start*, se encarga de recoger los datos introducidos en la interfaz, para después utilizar las funciones del módulo de procesamiento, y comprobar que los datos introducidos son correctos. Estos datos se almacenan en ficheros Pickle[6], para después enviarlos a la plataforma. En total, se han creado dos ficheros: uno para configurar el experimento y otro para configurar la plataforma.

Se ha decidido el uso de la librería Pickle, Python object serialization, porque serializa la entrada que se le entrega a la función de escritura, transformándola en un array de bytes, y deserializandola al leerla pasando de un array de bytes al conjunto de los datos originales, sin necesidad de realizar ninguna transformación, ni casting a los datos, lo que resulta perfecto para este propósito. Para realizar la lectura de estos ficheros en la plataforma, se ha implementado la función: *leer_ ficheros_graphicPyHuele*, que recibe sendos ficheros ya tratados, los lee y devuelve los diccionarios para proceder a la captura de los datos.

[5]https://docs.python.org/3/library/tk.html
[6]https://docs.python.org/3/library/pickle.html

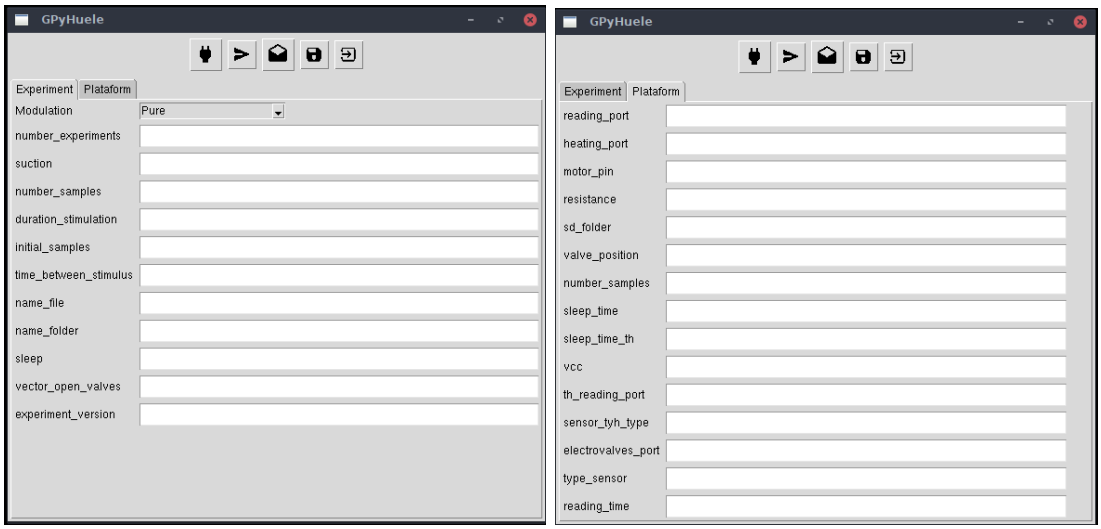


Figura 4.4: Figura de las pestañas que componen la interfaz gráfica. A la izquierda se muestra la pestaña que permite la introducción de los parámetros de configuración el experimento y a la derecha los que permiten la configuración de la plataforma.

Figura 4.5: Imagen de los botones que realizan las acciones de recoger los datos de para la conexión ssh, comenzar el experimento, cargar, guardar y salir en ese orden

La siguiente clase implementada, de la interfaz gráfica, es GPyHuele_Controller, y se encarga de recibir y gestionar los eventos producidos en la interfaz y de enlazar la interfaz con la lógica del programa.

La última clase que se ha implementado es GPyHuele_Model, y se encarga de lanzar los experimentos. Para conectarse de forma remota a la plataforma y lanzar los experimentos, se ha utilizado el módulo de Python Paramiko[7], que trabaja sobre el protocolo de encriptación ssh versión 2. Con esta librería de Python, y los datos de la conexión que recibimos de la clase GPyHuele_Controller, establecemos la conexión creando un canal de comunicación seguro, mediante la función *connect* de paramiko, y se envían por ese canal los ficheros con los datos de configuración, tanto para la plataforma como para las capturas que se quieren realizar. Por último, ejecutamos remotamente el comando de inicio de la captura, usando la función *exec_command*, y cerramos la conexión, con la función *close*, enviando un mensaje al usuario de que todo ha funcionado correctamente. Como el programa de ejecución, PyHuele, se desvincula de la terminal de ejecución él solo, no es necesario introducir ningún comando adicional, y se puede cerrar la conexión sin preocuparse de que se finalice la captura. Todo este proceso se realiza a través de la función: *get_nose_data*.

Si en algún punto de este proceso ocurre algún error, se lanza un pop-up de la clase Tkinter para informar del error ocurrido al usuario y se detiene la ejecución.

4.2.5. Extracción de características mediante el uso del bulbo olfativo

En esta sección se procede a explicar como se ha implementado este método de extracción de características, dividiendo en dos partes la explicación: una para la obtención de las señales

[7] http://www.paramiko.org/

neuronales y la otra para la obtención de las características a partir de las señales neuronales.

4.2.5.1. Obtención de señales neuronales

Para generar las señales neuronales, utilizando la red biológica explicada en la sección 2.7.2, se ha implementado la función: *generar_spikes*. Esta función consta de un bucle que se encarga de generar los diferentes puntos que forman las señales. El total de iteraciones que realiza el bucle viene dado por el número de puntos de la señal original, que es la curva de respuesta al odorante obtenida del sensor, dividido por el paso de integración utilizado. Cada ejecución del bucle genera un punto para cada una de las diferentes neuronas de la red, utilizando para ello la función de integración escogida. Como método de integración se ha utilizado Runge-Kutta de orden 4. Este método calcula un total de cuatro sub-puntos auxiliares entre dos puntos de la señal, ver figura 4.6. Debido al uso de cuatro sub-puntos en la señal, este método es más preciso que otros integradores, como Euler, además de ser más rápido. El paso de integración indica cuantas unidades de tiempo nos desplazamos para obtener un nuevo punto. Como paso de integración se ha utilizado un valor de 0.01, lo que significa que el número de puntos de la señal neuronal resultante se verá multiplicado por un factor 100 con respecto a la original. A esta función, ver código 4.2, le pasamos, como argumento, la función que queremos integrar, el potencial de membrana y la variable de recuperación del potencial de membrana, junto con los diferentes argumentos que necesitan, devolviendo el siguiente valor de la serie. Tanto los valores del potencial de membrana, como la variable de recuperación obtenidos se van almacenando en listas hasta la finalización del bucle y se guardan en un fichero.

```
def RK(dt, k, x, f, *args):

    k1 = f(k*dt,x,*args)
    k2 = f((k+1/2)*dt, x + (dt/2)*k1,*args)
    k3 = f((k+1/2)*dt, x + (dt/2)*k2,*args)
    k4 = f(k*dt, x + dt*k3,*args)

    return x + (k1 + 2*k2 + 2*k3 + k4)*(dt/6)
```

Listing 4.2: Rutina de integración Runge-Kutta de orden 4 utilizada para la obtención de las señales neuronales.

Para comprobar si se ha producido un spike, simplemente se comprueba el valor devuelto por la función de integración para el potencial de membrana, y si se supera el valor de pico impuesto, v_{peak}, se reinicia el valor mediante la función de reset.

4.2.5.2. Extracción de características RQA de las señales neuronales

Para la extracción de las características RQA de las señales neuronales se debe generar el *recurrence plot* (RP) asociado a la señal obtenida de la red biológica. Para obtener el RP, tal y como se explicó en la sección 2.7.2, hay que observar la trayectoria del espacio de fases de la señal, y contabilizar como cualquier recurrencia cuando el espacio de fases de la señal se acerca lo suficiente a un punto por el que ya se ha pasado previamente. La distancia ϵ, que es la que define el umbral utilizado para contabilizar la recurrencia, según la formula 2.10, tiene un valor de 0.05.

Para generar la matriz de recurrencia o RP, se utiliza la fórmula 2.10, en la que se obtiene la diferencia de cada punto de la serie con el resto de esta, para después binarizar la matriz calculada, generando la así la matriz de recurrencia. Según el valor que se le otorgue a ϵ, el

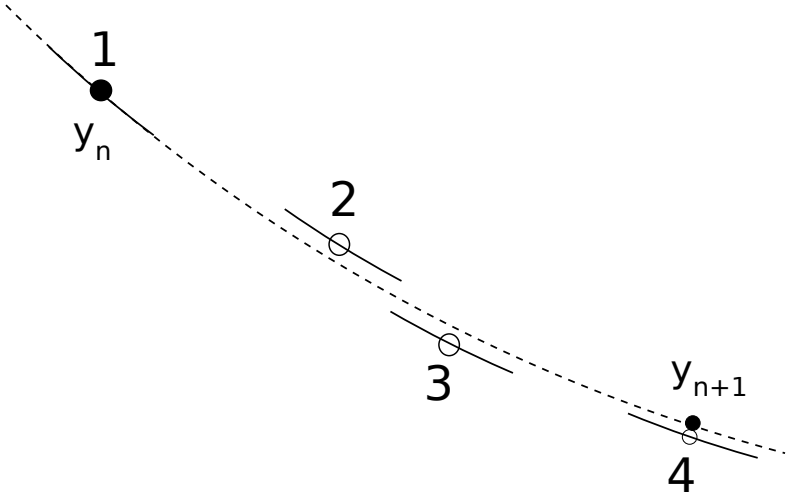


Figura 4.6: Imagen del cálculo del punto y_{n+1}. En cada paso se calculan cuatro puntos: el inicial, el final y dos intermedios. Imagen adaptada de (Press et al., 1996).

resultado obtenido del RP variará como se muestra en la figura 2.13, por lo que el valor que se le otorga a ϵ es importante en el cálculo de los RP.

El problema que surge al intentar aplicar este método radica en que las señales neuronales tienen un gran número de puntos, obteniendo así una matriz que por sus dimensiones no se puede almacenar en memoria. Debido a este problema, se tuvo que buscar otra manera de contabilizar las recurrencias del sistema.

Este problema se resolvió calculando las recurrencias de forma individual, tanto para las columnas como para las diagonales. Para las lineas verticales se fue restando cada elemento, de forma individual, a la serie obteniendo así una serie de distancias para cada elemento. Después, se aplica la fórmula 2.10, obteniendo una serie de unos y ceros. Por último, contamos las longitudes de los conjuntos de unos que se encuentran seguidos, y guardamos el resultado en una lista. El cálculo de las lineas diagonales fue un poco más complejo. Las diagonales de la matriz se calcularon restando la serie original consigo misma, desplazando una de las serie un elemento cada vez, siendo el resto del proceso igual que para las lineas verticales. La figura 4.7 muestra un ejemplo de este proceso. Una vez obtenidos estos valores, se puede calcular el vector de características RQA.

La realización de estos cálculos puede abarcar un tiempo bastante amplio, por lo que se tomó la decisión de paralelizar el cálculo de estas rutinas. Ya que los cálculos anteriormente mencionados no dependen unos de otros, se puede realizar una paralelización en forma de granja de tareas, reduciendo así el tiempo de ejecución, usando para ello la librería de Python Threading[8]. Mediante una paralelización en granja de tareas se obtiene el máximo rendimiento de los cores de la CPU, debido a que las tareas que ejecutan los hilos son independientes las unas de las otras. En estos casos, el rendimiento que se obtiene de cada hilo no es del 100 %, ya que se pierde un tiempo en la creación, gestión y cierre de los hilos, aunque ese tiempo se considera despreciable. Para realizar la paralelización del cálculo de la matriz de distancias, se han utilizado 4 hilos, ya que el procesador de la máquina que se ha utilizado consta de cuatro cores. Debido a la paralelización de las operaciones, ha sido necesario utilizar semáforos para poder guardar los datos en una lista sin sobre-escribirlos. La librería Threading también incluye la estructura de semáforos. Las funciones que se han desarrollado son las siguientes:

- **calcular_valores_verticales**: esta función recibe la señal completa y la divide en tantos

[8] https://docs.python.org/3/library/threading.html

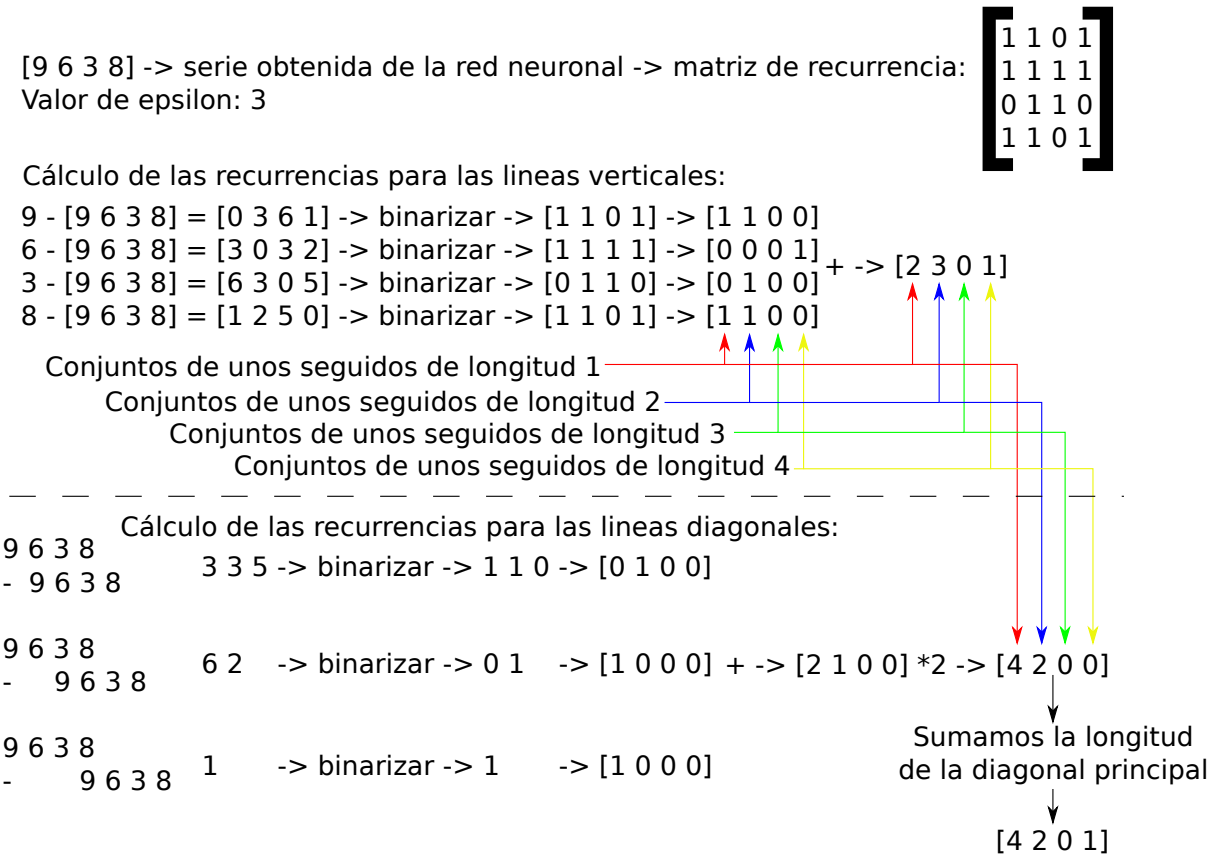


Figura 4.7: Ejemplo del proceso del cálculo de las recurrencias, tanto para las lineas verticales como para las diagonales.

trozos del mismo tamaño como hilos se empleen en el cálculo de la matriz de distancias. Tras crear los hilos y asignarles los trozos de la señal espera hasta que todos los hilos acaban.

- **calcular_valor_vertical**: esta función recibe un trozo de la serie neuronal y se encarga de realizar las operaciones, anteriormente expuestas, para obtener los valores de las columnas.
- **calcular_valores_diagonales**: esta función recibe la señal completa y la divide en tantos trozos del mismo tamaño como hilos se empleen en el cálculo de la matriz de distancias. Tras crear los hilos y asignarles los trozos de la señal espera hasta que todos los hilos acaban.
- **calcular_valor_diagonal**: esta función recibe un trozo de la serie neuronal y se encarga de realizar las operaciones, anteriormente expuestas, para obtener los valores en diagonal.

$$RR = \frac{1}{N^2} * (\sum_{i,j=0}^{N/4} R(i,j) + \sum_{i,j=N/4}^{2N/4} R(i,j) + \sum_{i,j=2N/4}^{3N/4} R(i,j) + \sum_{i,j=3N/4}^{N} R(i,j)). \quad (4.1)$$

Para el cálculo de las características RQA sobre el RP obtenido, se han creado tantas funciones como características se han calculado, teniendo un total de 9 características más la etiqueta de la clase. Al haber tenido que dividir el cálculo de los RP en 4 hilos diferentes, el cálculo de estas características se hace de forma individual en cada hilo, para luego sumar todos

los resultados parciales obteniendo el resultado final. En la fórmula, 4.1, se muestra un ejemplo de como se realiza este proceso para una de las características, pero el proceso es idéntico para todas. Las funciones son las siguientes :

- **recurrence_rate**: consiste en realizar un sumatorio de todos los puntos del RP y dividirlo entre el total de puntos:

$$RR = \frac{1}{N^2} * \sum_{i,j=0}^{N} R(i,j). \tag{4.2}$$

- **determinism**: se calcula mediante el cociente del sumatorio de la distribución de frecuencias, P(l), para la lista de longitudes de las lineas diagonales, por la lista de las longitudes de las lineas diagonales, l, a partir de un valor de longitud escogido, l_{min}, entre el sumatorio de la distribución de frecuencia de las longitudes de las lineas diagonales, por la lista de las longitudes de las lineas diagonales:

$$DET = \frac{\sum_{l=l_{min}}^{N} l * P(l)}{\sum_{l=1}^{N} l * P(l)} \tag{4.3}$$

- **laminarity**: consiste en realizar el mismo proceso que para el cálculo del determinism, pero utilizando el resultado obtenido para las lineas verticales en el RP:

$$LAM = \frac{\sum_{l=v_{min}}^{N} v * P(v)}{\sum_{v=1}^{N} v * P(v)}. \tag{4.4}$$

- **trapping_time**: se calcula mediante el cociente del sumatorio de la distribución de las frecuencias, P(v), para las longitudes de las lineas verticales, por la lista de longitudes, v, a partir de un determinado valor de longitud, v_{min}, entre el sumatorio de la distribución de frecuencia de la lista de longitud de las lineas verticales:

$$TT = \frac{\sum_{v=v_{min}}^{N} v * P(v)}{\sum_{v=1}^{N} P(v)}. \tag{4.5}$$

- **average_diagonal_length**: se calcula del mismo modo que el traping_time, pero utilizando los valores obtenidos para las lineas diagonales:

$$ADL = \frac{\sum_{l=l_{min}}^{N} l * P(l)}{\sum_{l=1}^{N} P(l)}. \tag{4.6}$$

- **entropy_diagonal_length**: se calcula aplicando la fórmula de la entropía de Shanon, utilizando la frecuencia de distribución de las longitudes para las lineas diagonales obtenidas de calcular el RP:

$$ENTR = -\sum_{l=l_{min}}^{N} p(l) ln(p(l)) \tag{4.7}$$

- **entropy_vertical_length**: se calcula aplicando la fórmula de la entropía de Shanon, utilizando la frecuencia de distribución de las longitudes para las lineas verticales obtenidas de calcular el RP:

$$ENTR = -\sum_{v=v_{min}}^{N} p(v) ln(p(v)). \tag{4.8}$$

- **max_vertical_length**: consiste en coger el valor máximo para los valores verticales obtenidos al calcular el RP.

- **max_diagonal_length**: consiste en coger el valor máximo para los valores diagonales obtenidos al calcular el RP.

5
Experimentos Realizados

En este capítulo se explican los diversos experimentos de captura realizados, junto con los scripts de ejecución.

5.1. Recolección de datos

Una parte del tiempo de realización de este TFM fue destinado a la captura de diversas muestras de datos para su posterior análisis y clasificación. Para poder realizar un análisis de de los datos con unas condiciones adecuadas, es necesario partir de una base de datos grande, por ello se han estado capturando datos de forma ininterrumpida, exceptuando el cambio de muestras, durante los primeros meses, para su posterior utilización. En total se han realizado un total de 330 capturas, obteniendo un total de 5280 transiciones, utilizando diferentes tendencias para su posterior clasificación.

La señal que se registra de un odorante difiere dependiendo del odorante que se estaba analizando previamente. Esto queda reflejado en el transitorio, que ocurre durante los primeros compases al realizar un cambio de un odorante a otro, ejerciendo una influencia en la señal que se registra antes de que se llegue a estabilizar. Al obtener respuestas diferentes, se puede llegar a diferenciar los odorantes, englobándolos en una misma clase, si se desea estudiar los odorantes, o englobándolos en diferentes clases, para estudiar las transiciones. Con esta base de datos se busca comprobar si mediante la modulación de temperatura se consigue diferenciar mejor los odorantes que se analizan, utilizando para ello la fase transitoria.

En los diferentes experimentos de captura realizados se han utilizado cuatro clases distintas de odorantes, los tres odorantes que se muestran en la figura 5.1 y el aire. Estos odorantes se diluyen para evitar que las señales que se registren superen el máximo admitido por la placa, dando lugar a la pérdida de información, así como la destrucción de la plataforma. Los odorantes, mostrados en la figura 5.1, se encuentran en su forma líquida, y se utiliza agua destilada para crear las diluciones. Para la captura de datos, se han utilizado una determinada muestra compuesta por los siguiente odorantes:

- Metanol
- Etanol

Odorante	Volumen(μl y ml)	Porcentaje
Metanol	125 μl - 0.125 ml	1.25 %
Etanol	250 μl - 0.25 ml	2.5 %
Butanol	100 μl -0.1 ml	1 %

Tabla 5.1: Porcentaje de odorante que compone la muestra utilizada para la captura de datos. El porcentaje es el total del volumen para cada bote.

- Butanol
- Aire

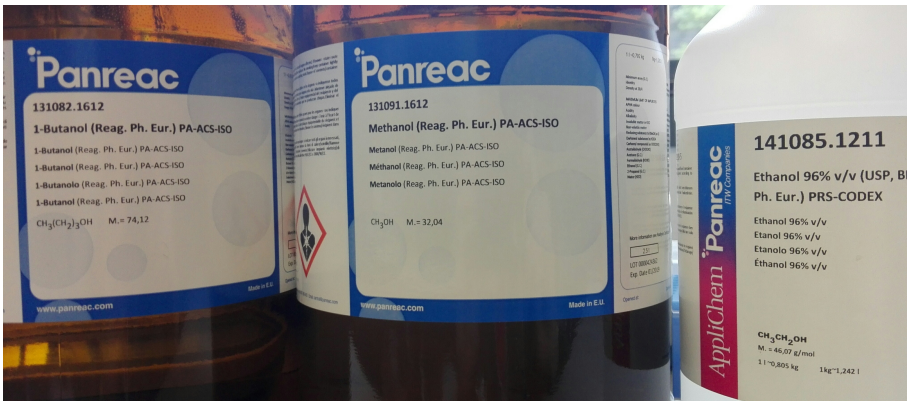


Figura 5.1: Frascos que contienen los odorantes en su estado líquido utilizados en la creación de las muestras.

Cada odorante se introduce en un bote diferente, mostrados en la figura 5.2. En cada bote, uno para cada odorante, se introducen 10 mL de dilución. De esos 10 mL, una parte se corresponde con el odorante y el resto con agua destilada. La tabla 5.1 muestra el porcentaje de odorante con respecto al total del volumen utilizado, siendo el resto agua destilada. Mediante el uso de esta dilución, se puede analizar los odorantes sin riesgo a que a la plataforma registre una señal superior a la permitida.

Figura 5.2: Imagen de un bote donde se almacenan los odorantes en su estado líquido.

Debido a la sensibilidad del sensor a los diferentes odorantes, se ha tenido que realizar una dilución diferente para cada odorante, de tal forma que aquellos a los que el sensor es más sensible necesitan una mayor cantidad de agua para evitar que la señal que se registre sature, y viceversa.

Para la realización de estas muestras es necesario el uso de instrumentos de medición con una alta calibración, mostrados en la imagen 5.3. Estos instrumentos son capaces de medir volúmenes del orden de micro-litros, para poder realizar la dilución lo más exacta posible. También es necesario el uso de dos vasos de precipitado, para contener en uno el odorante, en su forma líquida, y en el otro el agua destilada. Esto es importante ya que no se puede introducir la pipeta directamente en el frasco que contiene el odorante, ya que lo contaminaría. El proceso de creación de la muestra es el siguiente: para cada uno de los odorantes que se van a utilizar, se introduce la cantidad especifica valiéndonos de una micro-pipeta, ajustándola de forma manual hasta seleccionar el volumen de odorante deseado. Después se diluye con agua hasta alcanzar un volumen de un mililitro, para después utilizar la pipeta y completar los 10 mililitros totales.

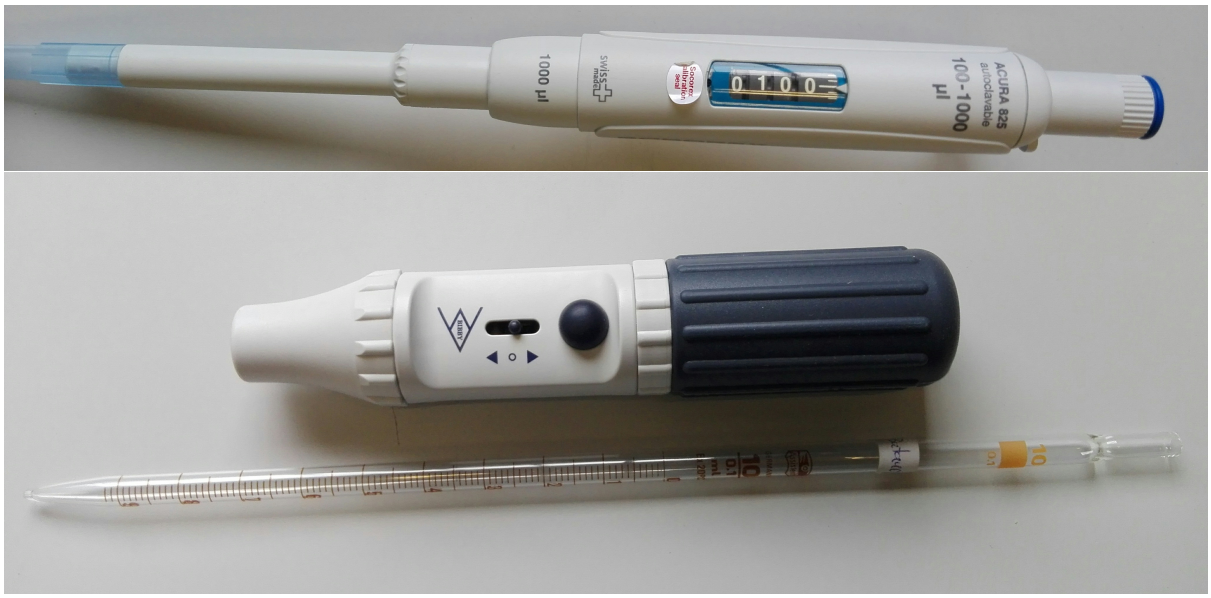


Figura 5.3: Instrumentos de medición utilizados en la creación de las muestras. La imagen superior se corresponde con la micro-pipeta, capaz de medir en un rango de un micro-litro y la imagen inferior se corresponde con una pipeta capaz de medir un volumen de hasta 10 mililitros.

Las muestras que se utilizan para la obtención de los datos tienen un ratio de uso. Según se utilizan, en los diferentes botes, el odorante contenido se va evaporando debido al uso y la señal que se va obteniendo se va degradando, acorde a este gasto. Debido a este gasto, que se produce en la muestra, es necesario cambiar los botes cada pocos días, para que la señal que obtengamos al analizar la muestra no se encuentre demasiado degradada, teniendo que descartarla debido a que no es posible analizarla correctamente. Este proceso se ha realizado cada dos o tres días, aproximadamente, durante el periodo de captura que abarcó el primer cuatrimestre del curso.

5.2. Experimentos

Los experimentos realizados consisten en registrar las señales de los diferentes odorantes seleccionados, en un determinado orden. Mediante el uso de las técnicas de modulación, se pretende comprobar si se pueden diferenciar los odorantes mejor que si utilizamos un algoritmo que tenga una temperatura de captura constante. Los algoritmos de modulación que se han utilizado han sido solamente dos:

- Captura de datos sin modulación.

- Captura de datos mediante temperatura variable y codificación temporal en amplitud.

Solo se han utilizado dos algoritmos de los cuatro implementados en la plataforma debido a que solamente se disponía de una plataforma operativa, y la plataforma solo disponía de un único sensor, por lo que solamente se ha podido utilizar un algoritmo de captura a la vez, no teniendo tiempo para realizar experimentos con otros algoritmos, a pesar de que en el software de control se tienen implementados todos los algoritmos de captura expuestos en la sección 2.5.

Aún habiendo capturado datos solamente con dos algoritmos de modulación podemos realizar un análisis de los datos capturados, y verificar si mediante la modulación con temperatura variable y codificación temporal en amplitud, se es capaz de diferenciar mejor los odorantes analizados que sin aplicar ninguna modulación para la mayoría de los casos que se estudien. Los parámetros utilizados para registrar los datos de los odorantes son los siguientes:

- number_experiments: 61
- suction: 70*
- duration_stimulation: 3600,90*
- number_samples: 1
- initial_samples: 30*
- time_between_stimulus: 0*
- name_file: sini,regresion*
- name_folder: Sini,Regresion*
- experiment_version: 2,1*
- modulation: 2
- tendency: 0.5,0(15),1(15),2(15),3(15)
- heat_sensor: 50*

Se han realizado un total de 61 capturas. La primera de ellas consiste en el análisis de un odorante aleatorio, para que el sensor se caliente y limpie el mayor número de impurezas que tiene en su superficie, durante un total de 3600 segundos.

Para el resto de las capturas realizadas, 60 en total, se ha dejado un total de 90 segundos para cada odorante. Este tiempo se considera suficiente para que el sensor capte el estímulo que produce el odorante, generando así una respuesta transitoria y llegando a un estado estable.

Como muestras iniciales se han capturado un total de 30 muestras para todos los algoritmos. Estas muestras son utilizadas por el algoritmo de regresión en los primeros compases de la captura de datos, para tener un conjunto de datos sobre el que aplicar la regresión. Este valor también define el tamaño de ventana utilizado, sobre los datos, para calcular la pendiente de la regresión. Este tamaño de ventana es lo suficientemente holgado para la obtención de una señal, clara y libre de ruido, ya que para ventanas pequeñas el algoritmo no tiene suficiente memoria de los datos captados, presentando un comportamiento errático, dando lugar a una señal muy ruidosa, y si se utiliza una ventana más grande puede dar lugar a que la eficacia del algoritmo se reduzca, debido que pueden utilizarse varias transiciones en el cálculo de la regresión.

Los valores de la tendencia definen la amplitud de la variación que se produce en la señal, como se explicó en la sección 2.4. Los valores de tendencia utilizados en los experimentos han sido: 0, 1, 2 y 3. Se han elegido estos valores, de entre todos lo posibles, debido a que se cubre un rango de tendencias aceptable, para realizar un primer análisis de los datos que se capturen. Con el valor de tendencia 0 no se aplica ningún cambio en la señal, utilizando el algoritmo de captura sin modulación. Mediante el uso de tendencias mayores se pretende realizar una variación en la señal con diferentes grados de amplitud, con el objetivo de intentar discernir los diferentes odorantes implicados. No se ha decidido utilizar valores mayores para la tendencia, debido a que la señal acabaría dando bandazos entre los límites, sin llegar a proporcionar ninguna información útil.

Con cada valor de tendencia se han realizado un total de 15 capturas, de modo que se puedan realizar comparaciones entre las señales captadas para un mismo experimento.

La succión del motor también influye en el proceso de captura, ya que si al motor de succión se le aplica un voltaje alto, succionará con más potencia y al sensor le llegará más odorante en menos tiempo, obteniendo así unas curvas de respuesta diferentes. Se ha utilizado una succión del 70 % sobre el máximo permitido, que son 5 voltios. Se ha decidido por el uso de esa potencia de succión porque hubo un problema durante la realización del TFG (Ortega, 2017), en el cual el motor se estropeó por usarse a la potencia máxima, por lo que se pasó a usar este valor.

Los odorantes que se han utilizado, junto con los porcentajes de dilución, son los que se han expuesto anteriormente en la tabla 5.1, y el orden en que se han presentado los odorantes al sensor es el siguiente:

- Muestras iniciales (Aire), Metanol, Metanol, Etanol, Etanol, Butanol, Butanol, Aire, Aire, Etanol, Metanol, Butanol, Metanol, Aire, Butanol, Etanol, Aire y Metanol.

En total se analizan una secuencia de 17 transiciones, de los cuales el primer odorante analizado lo confortan las muestras iniciales, formadas por aire, y el resto de odorantes forman las 16 transiciones que se quieren analizar posteriormente.

La frecuencia de captura utilizada para todas las capturas ha sido 1 Hz, o lo que es lo mismo, una muestra por segundo. El sensor utilizado durante las capturas ha sido siempre el mismo, ya que si en algún momento se hubiese cambiado de sensor, los efectos del drift variarían la señal obteniendo unas curvas diferentes.

El rango de variación de la temperatura del sensor se encuentra entre el 10 % y el 90 % del voltaje máximo que se aplica al sensor, que son 5V. Se ha escogido como temperatura inicial el 50 % del total, 2'5V, para ver el desarrollo de la señal, y evitar que la señal que la señal llegue a los limites del voltaje permitido al iniciar la captura. El valor de *time_between_stimulus* es 0, porque no queremos la presencia de aire en la secuencia de odorantes, ya que queremos analizar las transiciones seguidas. Los parámetros restantes, *name_file* y *name_folder*, definen el nombre de los ficheros y de la carpeta donde se guardan los ficheros con los datos captados.

Es necesario resaltar que para la captura de datos con tendencia 0 hubo ciertos problemas, debido a que las señales capturadas diferían mucho del patrón de captura del resto de valores para la misma tendencia, a la hora de capturar datos, y los datos capturados no se pudieron utilizar para la aplicación de la técnica DS, por lo que para la tendencia 0 no hay resultados en la clasificación de los datos mediante la aplicación de la técnica DS. Esta pérdida de datos fue puntual y no se repitió con ningún otro conjunto de datos.

6
Resultados

En este capítulo se exponen las diferentes técnicas aplicadas a los datos obtenidos. También se muestran los resultados de clasificar los datos, aplicando para ello técnicas de machine learning.

6.1. Bases de datos

Para la aplicación de las técnicas explicadas en el capitulo 2, se han utilizado dos bases de datos. La primera ellas, se ha llamado BD-EN1, y está compuesta por los datos captados durante la realización de este TFM. Como se comentó anteriormente, durante la creación de esta base de datos hubo ciertos fallos al usar el valor de tendencia 0, teniendo que descartar los datos captados, e impidiendo la aplicación de la técnica DS, para ese conjunto de datos. Este fallo fue puntual y no se repitió para ningún otro conjunto de datos. La segunda base de datos que se ha utilizado, llamada BD-EN2, es otra propia que tiene el GNB y que contiene datos sobre odorantes captados por un array de sensores de diversos modelos, junto con la temperatura y humedad ambientales. Esta base de datos contiene datos capturados en un entorno abierto, que abarcan un periodo de experimentación de un año aproximadamente. Esta forma parte de la tesis en curso: Exploración y caracterización de entornos con incertidumbre mediante sensores multimodales de Carlos García Saura.

Debido a la frecuencia de captura de la temperatura y humedad ambientales utilizada, y al corto periodo de captura de los odorantes, no es posible aplicar el algoritmo de decorrelación propuesto en (Huerta et al., 2016) a la base datos BD-EN1. La base de datos BD-EN2 cuenta con un total de 13 sensores, que han estado capturando un conjunto de datos de forma ininterrumpida durante un periodo de año aproximadamente, a excepción de cortos periodos de tiempo donde por fallos en la nariz electrónica se dejó de capturar datos. Esta pérdida de datos no es homogénea para todos los sensores, habiendo pérdidas mayores en un determinado conjunto de sensores. Aun así, esta base de datos proporciona los suficientes datos para la aplicación del algoritmo de decorrelación sin problemas.

La base de datos generada durante la realización de este TFM, BD-EN1, está compuesta por las señales obtenidas del análisis de diferentes muestras, creadas para este propósito. Estas muestras se han ido renovando regularmente cada 3-4 días, para evitar su completo desgaste,

y el descarte de las señales obtenidas. A esta base de datos de odorantes obtenida durante la realización de este TFM, BD-EN1, se le han aplicado varias técnicas para extraer características y ver si se pueden obtener buenos resultados, reduciendo el espacio de características de la señal original, acelerando el proceso de clasificación. Las técnicas utilizadas han sido:

- Exponential Movement Average (EMA), según lo explicado en la sección 2.7.1.
- Extracción de características mediante técnicas RQA de las señales neuronales obtenidas del modelo de bulbo olfativo de pequeños mamíferos, según lo explicado en la sección 2.7.2.

Además de utilizar las técnicas de extracción de características, se han clasificado las respuestas de los 4 odorantes incluyendo el transitorio, ver sección 5.1, tal y como son captados por el sensor sin aplicarles ningún tratamiento, para ver si mediante la extracción de características se obtiene buenos resultados, o en caso contrario, los resultados obtenidos son peores a los de las transiciones completas.

6.2. Decorrelación de la temperatura y humedad ambientales

Mediante la decorrelación de la temperatura y humedad ambiental se pretende eliminar los efectos producidos, por estos fenómenos atmosféricos, sobre los sensores utilizados, mediante la aplicación del modelo propuesto en (Huerta et al., 2016).

Para realizar la decorrelación de los datos, lo primero que se ha hecho ha sido un down-sampling de las muestras, realizando la media de un conjunto de puntos para todas las señales de BD-EN2. Este down-sampling se ha realizado agrupando los datos por tiempo, de tal forma que, cada vez que se contabiliza 60 segundos, en las series de datos originales, se realiza la media del número de muestras que se hayan agrupado, que no tiene porque ser el mismo para cada nueva muestra. De este modo se obtiene una muestra nueva, que es la que se utilizará.

Para aplicar la decorrelación de los datos se utiliza la fórmula 6.1.

$$R_i^* = R_i(t) - R_i(t-1)e^{\beta_1 \Delta H + \beta_2 \Delta H^2 + \beta_3 \Delta T_E \Delta H}. \tag{6.1}$$

Los parámetros β son aquellos que deben ajustarse al conjunto de datos que se use de entrenamiento. El valor de R_i^* representa el valor de la resistencia del sensor tras eliminar los efectos de la temperatura y humedad ambientales. Los valores de ΔH y ΔT_E se corresponden con la diferencia del tiempo t y del tiempo t-1 para la humedad y la temperatura exterior. Para realizar el ajuste del modelo, a los datos de entrenamiento, se ha utilizado la ecuación 6.2, donde los valores de R_I y R_F se corresponden con la resistencia del sensor en tiempo t-1 y en tiempo t respectivamente.

$$ln(\frac{R_F}{R_I}) = \beta_1 * \Delta H + \beta_2 * \Delta H^2 + \beta_3 * \Delta H * \Delta T_E. \tag{6.2}$$

Para ajustar los datos obtenidos al modelo, se ha utilizado una regresión lineal. Para entrenar esta regresión, y obtener los valores de las βs, se ha escogido un conjunto de datos correspondiente a un periodo aproximado de tres meses. El ajuste del modelo se ha realizado utilizando una regresión lineal. Debido a los fallos ocurridos en la nariz electrónica durante el registro de los datos, se ha escogido el sensor que menos pérdidas tiene: TGS modelo 2602. Para probar el modelo se ha escogido un periodo de tiempo de aproximadamente un mes.

Una vez que se han ajustado los parámetros de modelo, se procede a separar de la señal de la resistencia del sensor el efecto producido por la humedad y la temperatura ambientales, mediante la aplicación de la fórmula 6.1.

La imagen 6.1 muestra el aspecto de las señales de la temperatura y humedad ambientales normalizadas, que son comunes a todos los sensores de la base de datos BD-EN2, y la imagen 6.2 muestra la señal de la resistencia del sensor TGS modelo 2602. Tras aplicar la decorrelación de ambas señales, el resultado se muestra en la imagen 6.3. Como se puede ver, la temperatura exterior es el efecto ambiental que más variaciones produce en el sensor, de tal forma que cada vez que la temperatura exterior aumenta, la resistencia aumenta su valor y viceversa. Esto es lógico, ya que como se comentó en la sección 2.3.1, la temperatura del sensor influye en las reacciones de oxidación que ocurren en su superficie, afectando a los valores leídos.

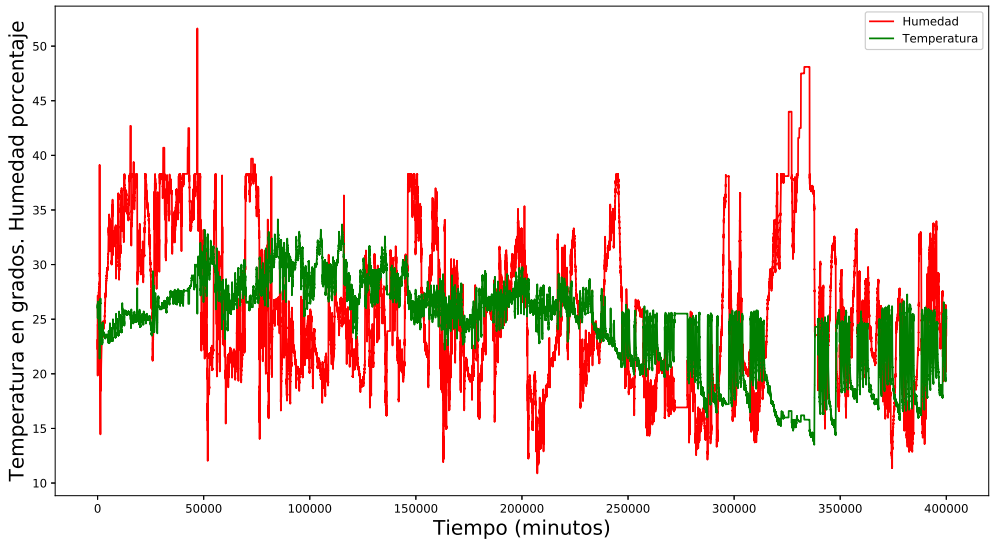


Figura 6.1: Señales correspondientes a la temperatura ambiental, en verde, y la humedad del ambiente, en rojo.

El objetivo de la aplicación de esta técnica es la decorrelación de las señales de la temperatura y humedad ambientales, no la clasificación de los datos. Según (Huerta et al., 2016), para obtener los mejores resultados al realizar la clasificación de los datos es necesario utilizar las señales puras, según se obtienen del sensor, y el conjunto de las mismas señales tras haberles aplicado este filtro.

6.3. Clasificación de los datos

La clasificación de los datos se llevado a cabo utilizando la primera base de datos, EN-BD1, que está constituida por los experimentos realizados durante la realización de este trabajo. Al conjunto de señales se les han aplicado unos métodos de extracción de características, con la esperanza de obtener unos resultados satisfactorios. Esta base de datos está compuesta por las transiciones de los diferentes odorantes utilizados para su creación, teniendo cada experimento realizado un total de 16 transiciones.

El objetivo definido ha sido la clasificación de las señales capturadas en los experimentos, utilizando dos métodos diferentes. Mediante estos dos enfoques se pretende comprobar si es

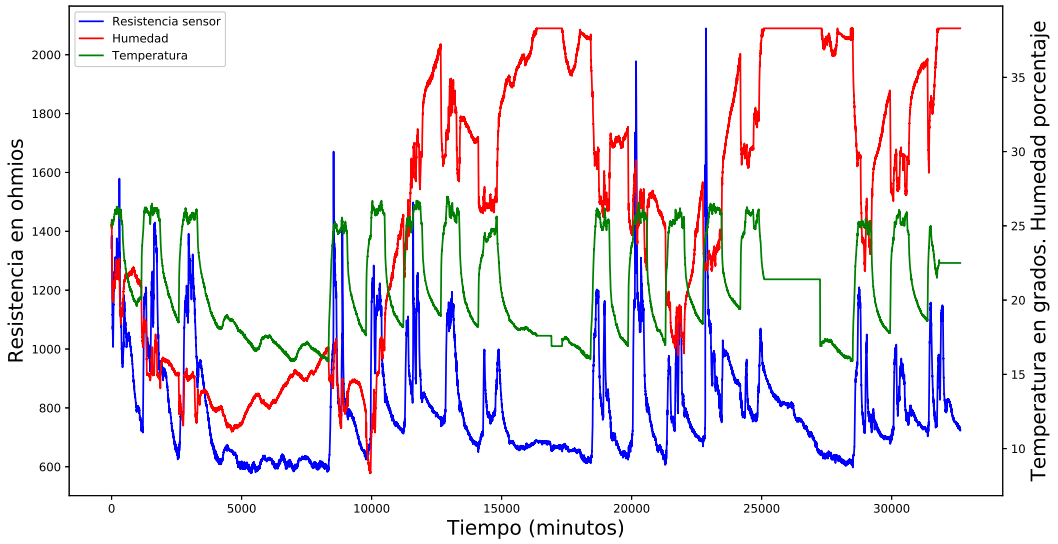


Figura 6.2: Figura que muestra las señales de la resistencia del sensor en azul, la humedad ambiental en rojo y la temperatura ambiental en verde, para el sensor TGS 2602 durante un periodo de un mes.

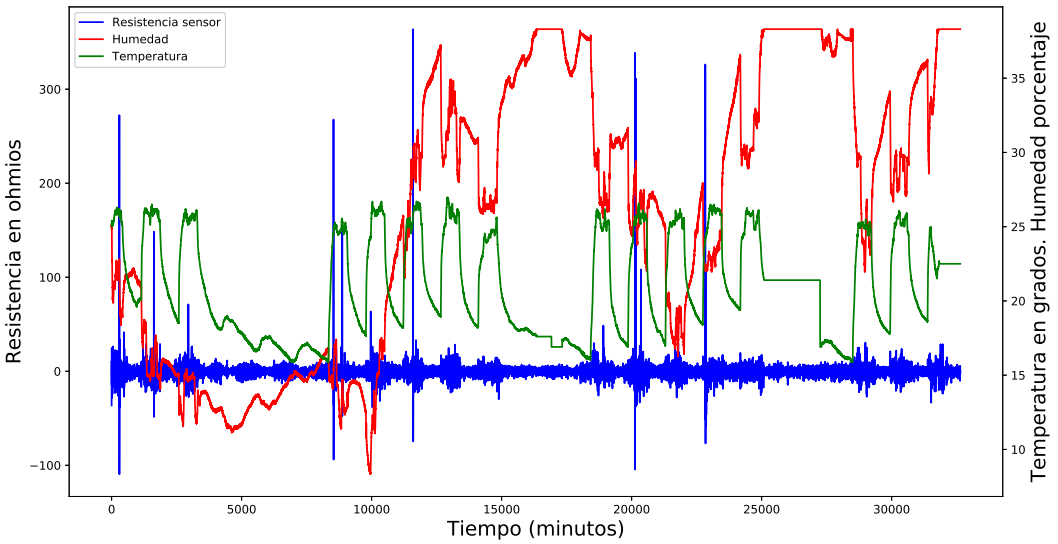


Figura 6.3: Señal de la resistencia del sensor en azul, humedad ambienta en rojo y temperatura ambiental en verde, después de eliminar los efectos de la temperatura y humedad ambientales utilizando la técnica de decorrelación.

posible diferenciar los odorantes implicados, incluyendo su fase transitoria, que ocurre durante el cambio de un odorante a otro, en un corto periodo de tiempo, evitando que la señal resultante pueda estabilizarse. Cada método se ha orientado a la clasificación de los odorantes de un modo distinto, con la intención de obtener el mejor. Para ello, el primer paso es dividir la señal completa en fragmentos, acordes a los odorantes y muestras iniciales, para después descartar las muestras iniciales y el primer odorante de la serie, debido a que no son de utilidad para este

propósito. Estos fragmentos son los que posteriormente van a clasificarse, de forma individual, dando a cada uno de ellos una clase según el enfoque que se aplique.

El primer método se ha orientado a la clasificación de los fragmentos obtenidos de las señales como odorantes. De este modo, se ha centrado el proceso de clasificación de cada fragmento en el odorante que se analizó en ese instante, incluyendo la fase transitoria que ocurre al cambiar de odorantes, como se explicó en la sección 5.1; de tal forma que si se transiciona, por ejemplo, al metanol desde cualquier otro odorante, la clase final es la correspondiente al metanol. Al centrar todo el proceso en los odorantes, se tiene un conjunto final de 4 clases diferentes.

El segundo método se ha orientado a la clasificación de los fragmentos de la señal, tratándolos como transiciones entre odorantes. En el proceso de clasificación se ha tenido en cuenta el odorante que se ha capturado previamente, centrándose en la transición del odorante A al odorante B, junto con el transitorio del odorante B, con la esperanza de obtener más información que utilizando solamente un odorante. Mediante este método se tiene un total de 16 clases distintas, haciendo referencia a cada una de las 16 transiciones utilizadas.

Para poder cumplir este objetivo y que el resultado sea convincente, es necesario tener una base de datos contundente. La base de datos utilizada, EN-BD1, consta de un total de 330 experimentos realizados con los diferentes algoritmos de captura implementados para distintos valores de tendencia, obteniendo un total de 5280 transiciones entre odorantes con una duración de 90 segundos cada una. Como se comentó en la sección 5.2, los datos utilizados fueron capturados con diferentes valores de tendencia. El motivo por el que se han utilizado diferentes valores de tendencia es comprobar si mediante la modulación se consiguen mejores resultados que utilizando una temperatura estable, y con que valor se obtiene el mejor resultado. Los valores empleados han sido: 0, 1, 2 y 3. La utilización de estos valores es suficiente para la captura, ya que para valores mayores la señal obtenida daría bandazos entre el límite superior e inferior.

6.3.1. Método de clasificación

La clasificación de los distintos datos se ha llevado a cabo utilizado máquinas de soporte vectorial (Support Vector Machines, SVM) (Vapnik, 2013). Como kernel utilizado en las SVM se ha usado un kernel gaussiano. Mediante el uso del kernel, la SVM transforma el espacio de características de las muestras a uno con mayor dimensión, separando las clases mediante el uso de un hiper-plano, para luego reducir esa dimensión obteniendo así una frontera de clasificación. El kernel utilizado afecta en gran medida a los resultados obtenidos de la SVM, ya que es el que realiza la transformación en la dimensión de los datos, por lo que hay que escoger un kernel adecuado a los datos que se estén tratando de clasificar.

Para demostrar que la modulación es efectiva frente al drift, se ha realizado una clasificación de los datos aplicando un paradigma de tiempo, en el cual el modelo de clasificación se entrena con los primeros datos captados, donde el efecto que produce el drift no afecta tanto a los datos, y se clasifican los últimos datos tomados, simulando así el ensuciamiento del sensor y donde el efecto del drift es más notorio. Con esto se pretende verificar si la modulación es capaz de mitigar el efecto producido por el drift.

El uso de SVM, como clasificador, implica la búsqueda de los parámetros óptimos C y γ, mediante los cuales la SVM obtiene los mejores resultados. El parámetro C define el coste de los casos no separables, es decir, define el grado de sobre-ajuste que tendrá el modelo. Con un valor muy alto el modelo perderá su capacidad de generalización y realizará un sobre-ajuste a los datos. Si el valor de C es bajo, el modelo no sobre-ajustará pero se clasificarán erróneamente muchos datos ya que generalizará demasiado. El parámetro γ es característico del kernel gaussiano y está relacionado con la anchura de la gaussiana. Para valores de γ altos la gaussiana tendrá una anchura pequeña y viceversa.

En el proceso de búsqueda de los valores C y γ óptimos, se ha seleccionado un sub-conjunto del total de los datos de los experimentos. La búsqueda de los valores óptimos ha supuesto la realización de dos búsquedas en malla, o grid. En la primera se barre un conjunto grande de valores, para luego centrar la búsqueda en el conjunto de valores para los cuales se obtiene un mejor resultado. La generación de los valores de búsqueda implica el uso de la función de Python: logspace. La malla de búsqueda de amplio espectro tiene un total de 13 valores que van desde 10^{-10} hasta 10^{10} para sendos parámetros. Una vez que se ha realizado la primera búsqueda y se han obtenido los mejores valores de C y γ, se realiza una segunda búsqueda en malla en la que se cubre un rango de valores más centrado en torno a aquellos valores cuyos resultados son los mejores para ambos parámetros, generando un total de 20 valores.

Una vez que se ha obtenido los mejores valores de C y γ para el SVM, se ha clasificado el conjunto total de los datos. Para ello se ha realizado una validación cruzada del total de los datos, dividiendo el conjunto de datos en 10 particiones idénticas y calculando la media de los resultados obtenidos. En el caso de los experimentos, en los que se ha aplicado el paradigma temporal, los datos se han clasificado directamente, sin realizar validación cruzada.

Se han escogido las máquinas de soporte vectorial para la clasificación frente a otras técnicas de machine learning, ya que debido al uso de los kernels, se puede transformar la dimensión original de los datos a una dimensión mayor, pudiendo encontrar de forma fácil la solución óptima para el conjunto de datos. Además en su espacio de búsqueda solo hay un mínimo global, con lo que se evita el caer en mínimos parciales, obteniendo siempre la solución óptima para el conjunto de datos y los parámetros especificados.

Durante la clasificación de los datos, se han generado varias figuras en las que se muestran de forma gráfica los resultados. Estas figuran se hayan en el anexo D, para facilitar la lectura del capítulo y evitar que las múltiples figuras saturen al lector. Estas figuras, que van desde la figura D.1 hasta la figura D.20, muestran los resultados de realizar las búsquedas paramétricas para los valores de C y γ, tanto para la búsqueda en amplio rango como para la centrada en el área donde se obtienen los mejores resultados. Junto a estas figuras también se pueden encontrar, desde la figura D.21 hasta la D.38, las matrices de confusión para las diferentes clasificaciones realizadas.

6.3.2. Calibración de los datos

Los datos capturados en los múltiples experimentos, han recibido un tratamiento antes de clasificarlos, con el fin de disminuir lo máximo posible o bien eliminar el efecto del drift en el sensor. La técnica utilizada para ello ha sido Direct Standardization, DS ver sección 2.6.1, ya que es la más simple de aplicar y se consiguen unos resultados aceptables, como se muestra en la figura 6.4. Para poder aplicar DS se ha utilizado como muestras de datos maestras las primeras capturas realizadas, cuando el sensor no se encontraba contaminado por la realización de múltiples experimentos. Como conjunto de datos esclavo se ha utilizado los últimos experimentos, ya que son aquellos en los que el sensor se encuentra más sucio por el uso. Para el cálculo de la matriz de transformación se han utilizado los valores de la resistencia del sensor, la temperatura que se le ha aplicado y el voltaje obtenido. Esta técnica de calibración se ha aplicado solamente a los datos obtenidos con el algoritmo de temperatura variable y codificación temporal en amplitud, debido a que la pérdida de datos sufrida con el algoritmo sin modulación, impide la aplicación de esta técnica. El uso combinado de esta técnica de calibración junto con la modulación de la temperatura, de calentamiento del sensor, busca mejorar los resultados obtenidos, disminuyendo el efecto del drift.

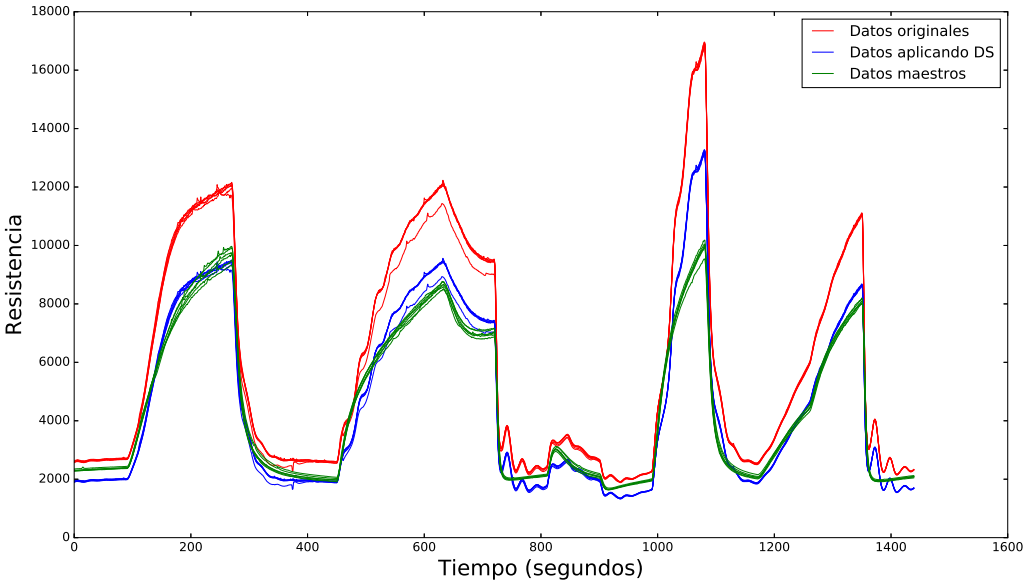


Figura 6.4: Imagen que muestra el efecto de la técnica DS sobre las series capturadas. Para cada color se han utilizado 6 series de datos, capturadas en un espacio de tiempo cercano. Las señales rojas muestran resistencias del sensor, en ohmios, al realizar la captura para una misma muestra de odorantes. El valor azul muestra esas mismas señales tras haberles aplicado DS y las señales en verde muestran aquellas seleccionadas como muestras maestras. Se aprecia como la diferencia que existe entre las señales se mitiga mediante el uso de esta técnica. Las señales representadas muestran la serie completa de odorantes que posteriormente se clasificarán.

6.3.3. Clasificación de los odorantes

En este apartado se exponen los resultados obtenidos al clasificar las señales, utilizando el primer método de clasificación explicado, donde se tratan las señales como odorantes, aplicando las diferentes técnicas de extracción de características.

6.3.3.1. Clasificación mediante el bulbo olfativo

La aplicación de este proceso ha supuesto la transformación de las curvas de respuesta de las diferentes transiciones de los odorantes, en señales neuronales para luego obtener características mediante RP y técnicas RQA.

Este método se ha aplicado al conjunto total de los datos, obteniendo las matrices de búsqueda de amplio espectro, que se muestran en la figura D.1, en forma de mapas de color. De izquierda a derecha y de arriba a abajo las tendencias utilizadas, en la figura, son: 0, 1, 2 y 3. Se puede observar que el rango de valores, para los cuales se obtienen los mejores resultados, son muy similares para todos los casos, concentrándose entre 1e0 y 1e10 para la C y desde 2.154e-7 a 2.154e3 para el valor de γ.

Las gráficas de la figura D.2 muestran, en el mismo orden, los resultados de realizar una búsqueda más centrada en aquellos valores obtenidos de la búsqueda anterior que aportan un mejor resultado para cada una de las tendencias. Los mejores valores de C y γ para cada una de las tendencias se muestran en la tabla 6.1, junto con los mejores aciertos obtenidos al clasificar. Los resultados obtenidos muestran que, para una tendencia de 3, los aciertos obtenidos son mejores que para cualquier otro valor de tendencia utilizado. Esto puede significar que para

	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
C	1.2742e6	1e6	5.4555e8	1.4384e6
γ	7.8475e-1	1.1288e-1	4.8329e-1	4.8329e-1
Porcentaje acierto	68.6	68.2	66.1	70.2

Tabla 6.1: Valores C de y γ para los cuales se obtienen los mejores resultados, junto con el porcentaje de acierto para las características obtenidas mediante el bulbo olfativo y cross validation.

	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
C	1.6237e8	1.8329e6	7.8475e6	1.6237e7
γ	7.8475e-8	3.3598e-7	2.3357e-8	7.8475e-9
Porcentaje acierto	94.86	93.04	91.09	89.59

Tabla 6.2: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando EMA y realizando cross validation.

valores de tendencia mayores de 3 los resultados pueden mejorar, o empeorar si la tendencia 3 es un máximo global en los resultados. El modelo neuronal utilizado requiere un mayor ajuste, debido a que los valores de conexión de las neuronas utilizados en el modelo son los mostrados en (Jing et al., 2016), pudiendo mejorar los resultados si se utilizan otros diferentes, desacoplando más las señales que retorna el bulbo olfativo.

6.3.3.2. Clasificación de datos mediante EMA

Mediante el uso de EMA (Exponential Movement Average) se busca obtener información de los periodos transitorios de las señales correspondientes a las transiciones de los odorantes.

Para cada transición entre odorantes se han generado tres señales EMA distintas, utilizando los siguientes valores para el parámetro α: 0.1, 0.01 y 0.001. En total, se ha utilizado un vector de 8 características para realizar la clasificación, como se explicó en la sección 2.7.1. Las características utilizadas son: el valor máximo y mínimo para cada serie EMA, obtenida a partir de los tres valores de α, y la diferencia entre el valor máximo y el valor mínimo de la resistencia (Vergara et al., 2012), tanto normalizado como sin normalizar. En la figura D.3, en el mismo orden que las imágenes mostradas con anterioridad, se puede ver el resultado de la búsqueda en malla de amplio rango para los valores de C y γ.

Los mejores resultados para esta primera búsqueda se concentran, para todos los mapas de calor, en torno a los valores de 2.154e3 a 1e10 para el parámetro C y de 1e-10 a 2.154e-2 para γ. En la figura D.4, se muestran los resultados de la búsqueda en malla para los valores centrados en torno a aquellos de los que se han obtenido un mejor resultado. Los mejores valores de C y γ para cada una de las tendencias se muestran en la tabla 6.2, junto con el mejor resultado obtenido.

Para los valores de tendencia mayores que 0 hemos repetido estos pasos aplicando previamente la técnica de calibración: DS. En las figuras D.5 y D.6 se muestran los resultados de las búsquedas en malla, para la obtención de los mejores valores de C y γ, y en la tabla 6.3 se pueden apreciar los valores para los cuales se obtiene un mejor resultado. Se puede ver, en las figuras D.5 y D.6, que los mejores valores de C se encuentran desplazados a la derecha, en comparación a la búsqueda realizada sin aplicar DS, entre los valores 1e-5 y 1e3.

Los resultados obtenidos mediante el uso de la técnica EMA son satisfactorios, convirtiéndola en una buena opción para la extracción de características, obteniendo los mejores resultados utilizando el algoritmo sin modulación, para la base de datos BD-EN1. La aplicación de la

	Tendencia 1	Tendencia 2	Tendencia 3
C	100.	48.3293	2.6366e6
γ	37.9269	5.4555	1.8329e-1
Porcentaje acierto	95.54	93.93	92.85

Tabla 6.3: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando EMA y habiendo aplicado DS y cross validation.

técnica DS mejora los resultados obtenidos, al compararlos con aquellos a los que no se ha aplicado esta técnica. Aún así, no se puede afirmar que los resultados obtenidos, mediante la aplicación de DS sean mejores que los obtenidos sin aplicar esta técnica, ya que no hay una referencia para realizar una comparación debido a la pérdida de datos.

6.3.3.3. Clasificación de las señales puras

La clasificación de las señales puras se ha llevado a cabo separando los datos en varios conjuntos, para después clasificarlos. Los conjuntos de datos generados son:

- La resistencia del sensor.
- La temperatura de calentamiento del sensor.
- La resistencia del sensor aplicando la temporalidad a los datos.
- La resistencia del sensor aplicando la temporalidad a los datos y utilizando la técnica de extracción de características EMA.

Mediante la creación de estos conjuntos de datos se pretende comprobar si a través de alguna de las huellas obtenidas, con el uso de las diferentes modulaciones, se consiguen diferenciar mejor los odorantes. En los dos primeros conjuntos y en el último, también se han clasificado los datos con tendencias mayores que 0, tras haber aplicado previamente Direct Standadization, para calibrar las señales.

En los dos últimos conjuntos se busca aplicar la temporalidad de los datos a su clasificación, para ver si se obtienen buenos resultados en la supresión de los efectos causados por el drift. Para ello se ha clasificado la SVM utilizando los primeros experimentos realizados para entrenar, donde el sensor no se ha ensuciado por las continuadas capturas realizadas. El objetivo es comprobar si un modelo entrenado con datos obtenidos sin que el sensor se encuentre sucio por el uso es capaz de clasificar datos correctamente, ignorando los efectos producidos por el drift. Los datos utilizados se capturaron en un rango de tiempo cercano, es decir, los experimentos implicados en el entrenamiento se realizaron en un periodo de tiempo similar, al igual que los utilizados en la clasificación. Con esto nos aseguramos que el drift es el mismo, o muy similar, en todos los datos capturados.

6.3.3.3.1 Resistencia del sensor

Las figuras D.7 y D.8 muestran los mapas de calor con los resultados de las búsquedas de los valores de C y γ para las diferentes tendencias. Los resultados obtenidos de la primera búsqueda en grid se agrupan en torno a valores de 1 hasta 1e10, para la C y de 1e-10 a 2.154e-7, para la γ. En la tabla 6.4 se muestran los mejores valores obtenidos para C y γ en cada una de las tendencias, junto con el resultado para cada uno de los valores expuestos.

	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
C	6.1584e1	1e5	2.6366e7	1.1288e2
γ	2.6366e-9	1e-10	2.0691e-10	4.2813e-10
Porcentaje acierto	98.45	98.70	99.14	99.01

Tabla 6.4: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando solamente la resistencia del sensor y aplicando cross validation.

	Tendencia 1	Tendencia 2	Tendencia 3
C	1e5	1.8329e2	61.5848
γ	1.6237e-2	2.3357e-1	2.3357e-01
Porcentaje acierto	99.29	99.56	99.62

Tabla 6.5: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando solamente la resistencia del sensor habiendo aplicando DS a los datos y clasificando mediante cross validation.

Se puede observar que para cualquier valor de tendencia, el resultado conseguido en promedio, es mejor que para el algoritmo sin modular, consiguiendo el mejor resultado con un valor de tendencia de 2. Esto demuestra que, para esta base de datos, el uso de técnicas de modulación ayuda a la diferenciación de los odorantes.

En estos datos se ha aplicado DS, con la intención de comprobar si se consiguen mejorar los resultados. La figura D.9 muestra los resultados de la búsqueda en malla para un amplio rango de valores, y la figura D.10 muestra los resultados para valores más acotados, en torno a aquellos con los que se obtiene un mejor resultado.

Tras haber aplicado DS los resultados obtenidos, estos mejoran con respecto a aquellos donde no se ha aplicado esta técnica, tal y como se muestra en la tabla 6.5. Aun así, al no haber podido aplicar esta técnica a los datos obtenidos sin modular, debido a la pérdida de datos acaecida, no se puede afirmar que mediante el uso de técnicas de modulación los resultados mejoren.

6.3.3.3.2 Temperatura del sensor

Con la huella de la temperatura se ha seguido el mismo proceso utilizado para la resistencia, con el fin de comprobar si los resultados que se puedan obtener lleguen a ser mejores. En la figura D.11 se muestra la búsqueda en grid de grano grueso, mientras que en la figura D.12 se ha afinado la búsqueda de los valores C y γ, centrándola en aquellos donde se obtienen los mejores resultados. Se puede observar, para la búsqueda en amplio rango, que los valores límite coinciden con los obtenidos por la resistencia.

La tabla 6.6 muestra el resultado de realizar las búsquedas en malla, para los valores de C y γ, junto con los resultados obtenidos al clasificar utilizando sendos valores. Se puede observar que no se ha tenido en cuenta el algoritmo sin modular. Esto es debido a que la temperatura es siempre constante, la inicial, por lo que el acierto máximo siempre va a ser de un 25 %, para los odorantes y un 6,25 %, en el caso de las transiciones. El mejor resultado para la clasificación de la huella de la temperatura del sensor se consigue utilizando el valor de tendencia 1. Si se realiza una comparación de los resultados, aplicando la técnica de calibración, los resultados mejoran tal y como se muestran en la tabla 6.7, obteniendo el mejor resultado utilizando un valor de tendencia de 2.

	Tendencia 1	Tendencia 2	Tendencia 3
C	12742.749857	1e+6	3.7926e2
γ	1.6237e-3	3.3598e-7	6.1584e-5
Porcentaje acierto	98.51	95.75	97.74

Tabla 6.6: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando solamente la temperatura del sensor aplicando cross validation.

	Tendencia 1	Tendencia 2	Tendencia 3
C	695.1927	1438.4498	2.6366e6
γ	2.9763e-1	2.3357e-1	1.8329e-1
Porcentaje acierto	99.16	99.48	99.35

Tabla 6.7: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando solamente la temperatura del sensor aplicando la técnica DS a los datos y utilizando cross validation.

6.3.3.3.3 Resistencia del sensor aplicando la temporalidad a los datos

En el caso de la resistencia se ha realizado también una clasificación aplicando el paradigma de la temporalidad. Para ello se ha seleccionado un conjunto de las primeras capturas realizadas para entrenar los modelos, clasificando con las últimas capturas, buscando así el ensuciamiento del sensor para obtener datos donde el efecto que provoca el drift sea notorio.

La figuras D.15 y D.16 muestran los mapas de calor realizados, durante las búsquedas de los valores de C y γ, que aportan mejores resultados al clasificar. Los mejores valores de C se agrupan entre 10e-10 y 10e10, y los de γ se agrupan solamente entre 10e-10 y 2.154e-7 para tendencias de 0 y 1. Para tendencias de 2 y 3, los valores de γ, se agrupan entre 10e-10 y 4.642e-9.

En la tabla 6.8 se muestran los valores de C de y γ para los cuales se obtiene un mejor resultado, así como el acierto obtenido al clasificar. Durante la clasificación llevada a cabo no se ha realizado una validación cruzada, debido a que los conjuntos de datos utilizados para entrenar y validar son estáticos, y no pueden intercambiarse. Los resultados muestran que el uso de la técnica de modulación temporal en amplitud genera mejores resultados al clasificar, que utilizando el algoritmo sin modular, para la base de datos: BD-EN1, consiguiendo el mejor resultado para un valor de tendencia 3. A la vista de estos resultados puede afirmarse que el uso de los algoritmos de modulación de la temperatura consigue mitigar los efectos del drift.

6.3.3.3.4 Clasificación de la resistencia mediante EMA aplicando la temporalidad a los datos

Debido a los buenos resultados obtenidos mediante el uso de la técnica EMA, se ha decidido utilizarla para la clasificación de la resistencia, aplicando el paradigma de la temporalidad. Con esto se pretende comprobar si mediante EMA se pueden conseguir mejores resultados, en

	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
C	18.3298	78475.9970	2.9763e2	3.3598e7
γ	1e-8	1e-10	1.2742e-10	1.6237e-10
Porcentaje acierto	88.33	91.875	81.875	92.29

Tabla 6.8: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando la resistencia del sensor aplicando la temporalidad a los datos capturados.

	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
C	18.3298	78475.9970	2.9763e2	3.3598e7
γ	1e-8	1e-10	1.2742e-10	1.6237e-10
Porcentaje acierto	88.54	81.04	80	87.08

Tabla 6.9: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando EMA y aplicando la temporalidad a los datos capturados.

	Tendencia 1	Tendencia 2	Tendencia 3
C	78475.9970	2.9763e2	3.3598e7
γ	1e-10	1.2742e-10	1.6237e-10
Porcentaje acierto	86.45	81.66	81.25

Tabla 6.10: Valores C de y γ para los cuales se obtienen los mejores resultados utilizando la extracción de características con EMA, aplicando DS a los datos y aplicando la temporalidad a los datos capturados.

comparación a los obtenidos anteriormente, al clasificar las señales completas. Las figuras A y B muestran el proceso de búsqueda del mejor valor de C y γ, tanto para grano grueso como para grano fino. En la tabla 6.9 se muestran los mejores resultados obtenidos para cada una de las tendencias, así como los mejores valores de la C y γ.

Se ha decidido aplicar la técnica DS a los datos, para después aplicar el paradigma temporal a la clasificación con EMA. Las figuras D y E muestran las matrices de búsqueda, para los valores C y γ, con los cuales los resultados obtenidos son óptimos. La tabla 6.10 muestra el resultado de clasificar los datos, habiendo aplicado DS previamente, junto con los parámetros C y γ utilizados.

Los resultados muestran que con EMA se consiguen mejores resultados utilizando el algoritmo sin modular, al igual que con el conjunto de datos de la sección 6.3.3.2. El uso de la técnica DS mejora los resultados obtenidos, en comparación alos mostrados en la tabla 6.9, excepto para la tendencia 3, que empeora. Al igual que con los casos anteriores, no se puede afirmar que mediante DS los resultados que se consiguen sean mejores mediante el uso de técnicas de modulación, debido a la pérdida de datos ocurrida.

6.3.4. Clasificación de las transiciones

Como se comentó con anterioridad, el segundo método de clasificación aplicado consiste en el tratamiento de las señales como transiciones. Para ello cada señal capturada es dividida en fragmentos según los odorantes analizados. Esto supone 17 fragmentos más las muestras iniciales, las cuales se descartan junto con el primer odorante analizado, ya que no son de utilidad para este propósito. Cada uno de los fragmentos restantes supone una clase distinta, ya que para cada clase se tiene en cuenta, no solo el odorante al que se realiza la transición, sino también el que se ha capturado previamente, centrando el proceso de clasificación en el conjunto de ambos odorantes. En total se tienen 16 clases diferentes, cuatro veces más que en el método anterior, lo que a priori complica el problema de clasificar los datos.

El proceso de clasificación seguido es el mismo que el llevado a cabo con el primer enfoque, dividiendo los datos en los mismos grupos. Los resultados obtenidos se muestran en la tabla 6.11, mientras que en la tabla 6.12 se pueden ver los valores de C y γ óptimos utilizados al clasificar. Los resultados alcanzados son muy satisfactorios, obteniendo aciertos del 80 % y 90 %, siendo muy similares a los obtenidos aplicando el enfoque de los odorantes. La excepción la pone el bulbo olfativo, obteniendo unos aciertos que no superan el 41.25 % necesitando, como se comentó

Método utilizado	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
Bulbo olfativo	35.83	41.25	39.375	35
EMA	91.99	87.8	89.65	84.29
EMA - DS	-	96.07	96.09	93.87
Resistencia	96.89	98.19	99.05	98.78
Temperatura	NP	97.07	99.10	98.46
Resistencia - DS	-	98.95	99.39	99.35
Temperatura - DS	NP	99.06	99.25	99.24
Resistencia - Temporal	86.45	93.95	86.04	93.12
EMA - Temporal	72.29	80.83	77.5	69.79
EMA - DS - Temporal	-	87.70	82.70	77.70

Tabla 6.11: Tabla resumen de los resultados de realizar la clasificación de las transiciones obtenidas. Se puede observar que los resultados son bastantes buenos, teniendo en cuenta que el enfoque de las transiciones es más complejo que el de los odorantes. La anotación NP indica que no procede la clasificación de esos datos, debido a que los resultados siempre son los mismos. El símbolo: "-" indica que no se ha podido llevar la clasificación por la pérdida de datos acaecida. Temporal indica el uso del paradigma temporal en el proceso de clasificación, con el cual se entrena el clasificador utilizando los primeros datos capturados y se clasifican los últimos, sin aplicar ningún tipo de validación.

anteriormente, más investigación para mejorar el resultado.

Se puede observar, al igual que para el enfoque de los odorantes, que el uso de la modulación mejora el resultado obtenido utilizando el paradigma de la temporalidad, frente al algoritmo de captura sin modulación excepto en el caso del uso de EMA, los cuales empeoran al utilizar la modulación.

Método utilizado	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
Bulbo olfativo	C - 6.1584e7,γ - 1	C - 4.2813e6,γ - 1.2742e-2	C - 5.4555e6,γ - 2.6366e-2	C - 1000000,γ - 1.8329e-1
EMA	C - 2.3357e7,γ - 1.8329e-8	C - 7.8475e7,γ - 5.4555e-9	C - 5.4555e6,γ - 7.8475e-9	C - 6.9519e6,γ - 4.2813e-10
EMA - DS	-	C - 2.3357e7,γ - 1.8329e-1	C - 8.8586e6,γ - 1.2742	C - 1e7,γ - 1.1288e-1
Res./Vol.	C - 2.6366e4,γ - 3.7926e-9	C - 4.2813e3,γ - 3.3598e-10	C - 1.2742e6,γ - 8.8586e-10	C - 8.8586e3,γ - 4.2813e-10
Temp.	NP	C - 4.2813e6,γ - 2.6366e-7	C - 1e6,γ - 2.0691e-4	C - 2.0691e8,γ - 3.3598e-5
Res./Vol. - DS	-	C - 1.8329e3,γ - 4.2813e-2	C - 1.1288e5,γ - 5.4555e-2	C - 2.3357e6,γ - 1.4384e-1
Temp. - DS	NP	C - 1.6237e4,γ - 1.4384e-1	C - 1.8329e4,γ - 2.3357e-1	C - 2.3357e4,γ - 5.4555e-2
Res./Vol. - Temporal	C - 78.4759,γ - 4.8329e-9	C - 4.8329e2,γ - 1.1288e-9	C - 1e4,γ - 1e-10	C - 12.7427,γ - 2.6366e-10
EMA - Temporal	C - 8.8586e6,γ - 7.8475e-8	C - 1.8329e7,γ - 5.4555e-10	C - 4.2813e8,γ - 4.8329e-9	C - 4.8329e9,γ - 1.2742e-10
EMA - DS - Temporal	-	C - 8.8586e6,γ - 2.0691e-4	C - 8.8586e8,γ - 2.9763e-5	C - 2.6366e4,γ - 6.1584e-1

Tabla 6.12: Tabla resumen de los valores de C y γ con los que se obtienen mejores resultados al clasificar las transiciones. La anotación NP indica que no procede la clasificación de esos datos, debido a que los resultados siempre son los mismos. El símbolo: "-" indica que no se ha podido llevar la clasificación por la pérdida de datos acaecida. La abreviatura Res. hace referencia a la resistencia del sensor, Vol. hace referencia al voltaje y Temp. a la temperatura. Temporal indica el uso del paradigma temporal en el proceso de clasificación, con el cual se entrena el clasificador utilizando los primeros datos capturados y se clasifican los últimos, sin aplicar ningún tipo de validación.

6.3.5. Clasificación de los odorantes y las transiciones

En esta sección se realiza una comparación entre los resultados obtenidos en la clasificación entre odorantes y transiciones. La tabla 6.13 muestra un resumen de las características utilizadas en las clasificaciones, mostrando para cada método y valor de tendencia la siguiente información: total de patrones utilizados para la partición de test, total de patrones utilizados para la partición de train, si se ha realizado una validación cruzada con los datos y si se ha aplicado el paradigma de tiempo al clasificar los datos.

La tabla 6.14 muestra una comparativa de los resultados obtenidos al clasificar los datos utilizando ambos métodos. Se puede observar que los resultados obtenidos son muy similares, a pesar de que a priori el método de las transiciones al tener más clases debería tener un acierto menor. Esta similitud en los resultados puede deberse a que las señales utilizadas son facílmente diferenciables para el SVM, tras realizar la búsqueda de los valores óptimos de C y γ. De los dos enfoques utilizados, se consigue el mejor resultado utilizando el primero, donde se tratan las señales como odorantes, en la mayoría de los casos. Esto es debido a que al haber menos clases el SVM comete menos error al clasificar, siendo capaz de agrupar mejor las señales. Al clasificar los datos aplicando el paradigma de la temporalidad, los resultados mejoran utilizando el enfoque de las transiciones, siempre que no se aplique EMA. Esto es debido a que el clasificador no recibe toda la serie de datos donde puede apreciarse el ensuciamiento de las señales debido al drift, sino que se le entrena con los datos donde el drift es menos notorio, para clasificar donde es más notorio. Este cambio en las señales hace que el clasificador no sea capaz de diferenciar de forma tan precisa las señales, llegando a confundir los gases que son más similares entre sí. Al utilizar este método, el clasificador es entrenado con el mismo conjunto de señales, pero teniendo un mayor número de clases. Esto supone una mayor especialización, por lo que el clasificador es capz de diferenciar mejor los casos en los que el otro enfoque no es capaz de acertar.

En ambos casos, tanto sin aplicar la temporalidad como aplicándola, se puede decir que el uso de la modulación de la temperatura del sensor mejora los resultados obtenidos, a la hora de diferenciar los odorantes, excepto en el caso en que se utilice EMA, donde los resultados son mejores utilizando una captura sin modulación. Otra conclusión que se obtiene de la interpretación de los resultados es que la clasificación de los datos mediante el enfoque de los odorantes aporta mejores resultados si se clasifica toda la serie temporal de datos, mientras que si se desea clasificar datos aplicando un paradigma temporal, es mejor utilizar un enfoque orientado a las transiciones, donde se tengan en cuenta los odorantes que intervengan en el proceso, y no solo el odorante al que se transiciona.

	Odorantes con transitorio (4 clases) - Transiciones (16 clases)			
Método	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
Bulbo olfativo	384/3456/Sí/No	384/3456/Sí/No	384/3456/Sí/No	384/3456/Sí/No
EMA	384/3456/Sí/No	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
EMA - DS	-	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
Res./Vol.	384/3456/Sí/No	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
Temp.	NP	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
Res./Vol. - DS	-	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
Temp. - DS	NP	528/4752/Sí/No	528/4752/Sí/No	528/4752/Sí/No
Res./Vol. - Temporal	480/720/No/Sí	480/720/No/Sí	480/720/No/Sí	480/720/No/Sí
EMA - Temporal	480/720/No/Sí	480/720/No/Sí	480/720/No/Sí	480/720/No/Sí
EMA - DS - Temporal	-	480/720/No/Sí	480/720/No/Sí	480/720/No/Sí

Tabla 6.13: Tabla resumen de los resultados tras realizar la clasificación de los odorantes y las transiciones obtenidas. En cada celda se muestran las características que se han utilizado durante la clasificación. Para cada celda, de izquierda a derecha se muestra: número de muestras para una partición de test, el número de muestras para una partición de train, si se ha aplicado validación cruzada y si se ha aplicado el paradigma temporal a los datos. La anotación NP indica que no procede la clasificación de esos datos, debido a que los resultados siempre son los mismos. El símbolo: "-" indica que no se ha podido llevar la clasificación por la pérdida de datos acaecida. La abreviatura Res. hace referencia a la resistencia del sensor, Vol. hace referencia al voltaje y Temp. a la temperatura. Temporal indica el uso del paradigma temporal en el proceso de clasificación, con el cual se entrena el clasificador utilizando los primeros datos capturados y se clasifican los últimos, sin aplicar ningún tipo de validación.

	Odorantes con transitorio (4 clases)					Transiciones (16 clases)			
índ.	Método	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3	Sin modular	Tendencia 1	Tendencia 2	Tendencia 3
1	Bulbo olfativo	68.6	68.2	66.1	70.2	35.83	41.25	39.375	35
2	EMA	94.86	93.04	91.09	89.59	91.99	87.8	89.65	84.29
3	EMA - DS	-	95.54	93.93	92.85	-	96.07	96.09	93.87
4	Res./Vol.	98.45	98.70	**99.14**	99.01	96.89	98.19	99.05	98.78
5	Temp.	NP	98.51	95.75	97.74	NP	97.07	**99.10**	98.46
6	Res./Volt. - DS	-	99.29	99.56	99.62	-	98.95	99.39	99.35
7	Temp. - DS	NP	99.16	99.48	99.35	NP	99.06	99.25	99.24
8	Res./Vol. - Temporal	88.33	91.875	81.875	92.29	86.45	93.95	86.04	93.12
9	EMA - Temporal	88.54	81.04	80	87.08	72.29	80.83	77.5	69.79
10	EMA - DS - Temporal	-	86.45	81.66	81.25	-	87.70	82.70	77.70

Tabla 6.14: Tabla resumen de los resultados de realizar la clasificación de los odorantes y las transiciones obtenidas. La anotación NP indica que no procede la clasificación de esos datos, debido a que los resultados siempre son los mismos. El símbolo: "-" indica que no se ha podido llevar la clasificación por la pérdida de datos acaecida. La abreviatura Res. hace referencia a la resistencia del sensor, Vol. hace referencia al voltaje y Temp. a la temperatura. Temporal indica el uso del paradigma temporal en el proceso de clasificación, con el cual se entrena el clasificador utilizando los primeros datos capturados y se clasifican los últimos, sin aplicar ningún tipo de validación.

7

Conclusiones y trabajo futuro

7.1. Conclusiones

La finalización de este Trabajo de Fin de Máster ha supuesto la realización de varias tareas relacionadas con el ámbito de las narices artificiales. La primera de ellas ha supuesto la creación de una base de datos, BD-EN1, a través de la captura de odorantes mediante el uso de dos técnicas de modulación distintas, durante un periodo de tiempo de tres meses aproximadamente. El objetivo con el que se ha creado esta base de datos es verificar si, a través de las técnicas de modulación de la temperatura del sensor, es posible llegar a diferenciar mejor los odorantes que se analicen.

Los módulos software de Python han sido ampliados y mejorados, añadiendo nuevas funcionalidades que permiten ajustar varios parámetros de la plataforma que inicialmente eran estáticos, permitiendo así que la plataforma sea lo más versátil posible, incorporando nuevos comandos a la sintaxis del pseudo-lenguaje y cambiando por completo el tratamiento y análisis de los ficheros que recibe el software de experimentación. Junto con la ampliación y la mejora de los parámetros de la plataforma, se ha implementado un nuevo tipo de modulación que permite explorar el comportamiento del sensor en diferentes estados (Herrero-Carrón et al., 2015). Aprovechando todos estos cambios, realizados en el software de PyHuele, se ha saneado el código de la plataforma utilizando patrones de diseño software, junto con el uso de la orientación a objetos. Por último, se ha desarrollado e implementado una interfaz gráfica que permite el uso y conexión a la plataforma sin tener que conectarse utilizando para ello la terminal.

Como complemento añadido al software desarrollado se han estudiado y codificado varias técnicas para el tratamiento de los datos obtenidos en los experimentos. La supresión del efecto que provoca el drift en las señales se ha llevado a cabo estudiando e implementando la técnica Direct Standardization; mientras que para la eliminación de los efectos provocados por la humedad y la temperatura ambientales, se ha utilizado la técnica propuesta en (Huerta et al., 2016), utilizando para ello una base de datos propia del GNB, BD-EN2, con los suficientes datos para poder replicar el resultado del artículo (Huerta et al., 2016). La decorrelación de los efectos de la humedad y temperatura ambientales ha sido un éxito, consiguiendo replicar los resultados del trabajo (Huerta et al., 2016) a la perfección.

La clasificación de los datos se ha llevado a cabo utilizando Máquinas de Soporte Vecto-

riales, SVM, con un kernel gaussiano. Antes de proceder a la clasificación de los datos se han realizado dos búsquedas paramétricas, con el fin de obtener los parámetros C y γ con los que se consigue el mejor acierto al clasificar. Al clasificar los datos se han utilizado dos métodos diferentes. Inicialmente se ha clasificado el conjunto total de datos disponibles, con la intención de comprobar si mediante el uso de técnicas de modulación es posible mejorar la identificación de odorantes. Con el segundo método se ha utilizado un paradigma temporal, entrenando al clasificador con los primeros datos obtenidos en los experimentos, para después clasificar con los últimos. La finalidad de este método es el estudio del objetivo principal de forma más realista, ya que los clasificadores se entrenan con datos donde el drift no es tan notorio como en los datos que posteriormente se clasifican. En combinación con estos métodos se han aplicado dos enfoques diferentes: el primero consiste en clasificar los 4 odorantes incluyendo el transitorio de cada odorante, ver sección 5.1. El segundo método consiste en clasificar las transiciones de los odorantes, no solo centrándose en el transitorio del odorante, sino también en el odorante que se ha capturado con anterioridad, haciendo referencia al conjunto de ambos odorantes, habiendo un total de 16 clases posibles. Esto se muestra en la tabla 6.14, donde la parte izquierda muestra la clasificación de los odorantes, y la parte derecha muestra la clasificación de las transiciones.

Al clasificar los datos se han aplicado varías técnicas de obtención de características, implementadas durante el desarrollo de este trabajo, con el fin de clasificar los vectores de datos obtenidos. Además de esto se ha aplicado la técnica de calibración implementada Direct Standardization, con la que se han estandarizado varias señales para comprobar si es posible obtener mejores resultados que para señales a las que no se les ha aplicado ningún tratamiento.

Los resultados obtenidos del uso del bulbo olfativo y las técnicas RQA, de las curvas de respuesta de los odorantes, para el enfoque de los odorantes, ver tabla 6.14 fila 1, no son tan buenos como se esperaban que fuesen en comparación con los resultados de (Jing et al., 2016). Aun así, estos resultados no son tan malos como puedan llegarse a pensar, debido a que la plataforma de experimentación solo consta de un sensor, en vez de los 10 utilizados en el artículo; obteniendo los mejores resultados para un valor de tendencia de 3, tabla 6.14 fila 1, lo que puede indicar que si se aumenta la tendencia a valores más altos los resultados podrían ser mejores. Con el enfoque de las transiciones, ver tabla 6.14 fila 1, los resultados obtenidos tampoco son los esperados, ya que el mejor acierto conseguido es de 41.25 %, para un valor de tendencia de 1. Estos resultados se deben a la gran complejidad que implica el enfoque de las transiciones entre odorantes y, como para el enfoque de los odorantes, al menor número de sensores de los que dispone la plataforma. Estos resultados hacen que, por el momento, este método no sea muy buena opción para clasificar las transiciones. En ambos casos puede observarse que mediante el uso del algoritmo de modulación se consiguen mejores resultados que capturando datos con una temperatura constante. Los resultados obtenidos se pueden mejorar mediante la realización de un estudio completo de los parámetros que componen el modelo neuronal, ya que los que se han empleado se corresponden al artículo (Jing et al., 2016). De este modo se podría conseguir un desacople máximo en las señales neuronales generadas a través del modelo neuronal del bulbo olfativo, pudiendo ser un buen método a la hora de clasificar odorantes.

Con el uso de EMA los resultados obtenidos, tabla 6.14 fila 2, a través de la modulación de temperatura empeoran de forma nimia, frente a los obtenidos con una temperatura de captura estática. Aplicando la técnica de calibración Direct Standardization, los resultados mejoran, como se muestra en la tabla 6.14 fila 3, comparándolos con los obtenidos con EMA, tabla 6.14 fila 2. Esto no implica que aplicando DS se consigan mejores resultados a través de la modulación de la temperatura, ya que al no poder aplicar DS a los datos sin modular no se tiene una referencia. Aún obteniendo unos resultados que son débilmente peores con la modulación de la temperatura, estos son satisfactorios, convirtiendo a EMA en una buena opción para la extracción de características a la hora de clasificar datos.

La clasificación de las señales sin aplicar ningún método de extracción de características

aporta muy buenos resultados, mediante la modulación dinámica de la temperatura. Si se clasifica el conjunto total de los datos obtenido utilizando el enfoque de los odorantes, la mejor huella obtenida del sensor que puede utilizarse es la del voltaje/resistencia, habiendo aplicado la técnica de la modulación de la temperatura con codificación en amplitud, tabla 6.14 fila 4, para una tendencia de 2; mientras que si se decide utilizar el enfoque de las transiciones, el mejor resultado se obtiene con la huella de la temperatura de modulación que se aplicó al sensor durante la captura, tabla 6.14 fila 5, con un valor de tendencia de 2. Si en vez de clasificar el conjunto total de los datos, se aplica el paradigma temporal y se entrena con los primeros datos obtenidos, para luego clasificar con los últimos capturados, utilizando el enfoque de los odorantes el mejor resultado se consigue utilizando la huella del voltaje/resistencia, tabla 6.14 fila 8, con una tendencia de 3; mientras que con el enfoque de las transiciones, el mejor resultado se obtiene para la huella del voltaje/resistencia, tabla 6.14 fila 8, para una tendencia de 1. Al aplicar la técnica de calibración los resultados obtenidos mejoran, pero al igual que ocurre con EMA, tabla 6.14 fila 2, al no poder aplicar esta técnica a los datos capturados a temperatura constante, no puede afirmarse que se mitiguen los efectos provocados por el drift. De estos últimos resultados se desprende que mediante las técnicas de modulación empleadas se puede mitigar, en parte, el efecto que provoca el drift en los datos capturados. Este hecho puede verse en la tabla 6.14, en especial en la fila 8, donde se ha aplicado el paradigma temporal a la señal del voltaje/resistencia, junto con la modulación de la temperatura para diferentes valores de tendencia. En esta fila puede observarse con claridad que cuando se aplica la técnica de modulación, el error que obtiene el clasificador es mejor que cuando se captura con el algoritmo de temperatura estática. En la parte izquierda de la tabla, el mejor resultado se consigue con el algoritmo de modulación utilizando una tendencia de valor 3, y en la parte derecha con una tendencia de valor 1.

En global, la característica con la que se obtiene el mínimo error, para los experimentos realizados tanto aplicando el paradigma temporal como sin aplicarlo, es la huella de la resistencia/voltaje, obtenida a través de la aplicación del algoritmo de modulación de la temperatura del sensor, tal y como se muestra en la tabla 6.14, en las filas 4 y 8.

Las conclusiones obtenidas de la finalización de este trabajo son la demostración, en una primera aproximación, de que el uso de las técnicas de modulación generan resultados satisfactorios, para ciertos casos llevados a cabo en este trabajo, tal y como se muestra en la tabla 6.14; y que su uso combinado con la aplicación del paradigma temporal, explicado anteriormente parece que mitiga, en parte, el efecto que provoca el drift en las señales captadas por el sensor, aunque esto debe de estudiarse más a fondo.

7.2. Trabajo futuro

Con los resultados obtenidos se tienen varias líneas de investigación abiertas con las que se puede continuar investigando. Para empezar se pueden captar odorantes utilizando las técnicas de modulación restantes, que están implementadas en la plataforma. De este modo se puede comprobar si mediante su uso se consiguen unos resultados mejores a los obtenidos al clasificarlos, ya sea mediante la aplicación de técnicas de extracción de características o mediante la clasificación de las transiciones completas.

Otra tarea con la cual se puede continuar trabajando es la captura de datos mediante la técnica de modulación temporal en amplitud, afinando los parámetros de búsqueda basándose en los resultados expuestos en este trabajo, utilizando también las técnicas de extracción de características utilizadas.

Las capacidades de la interfaz gráfica se pueden ampliar, añadiendo alguna de las siguientes

mejoras: visualizar las capturas realizadas, visualizar los resultados según se van capturando a diferentes frecuencias o precargar un conjunto de valores por defecto al iniciar la interfaz, son algunas de las opciones posibles, entre las muchas posibles.

Se puede implementar el algoritmo de modulación propuesto por (Hossein-Babaei and Amini, 2014, 2012). En este método de experimentación se utiliza una variación brusca de la temperatura de calentamiento del sensor, para tratar de obtener una mayor cantidad de información acerca del odorante que se está analizando.

El método de extracción de características del bulbo olfativo necesita mucho más trabajo de investigación. Para empezar, se tiene que realizar una búsqueda paramétrica de los diferentes pesos que conforman la red. Con esta búsqueda se pretende obtener los mejores pesos de la red, para conseguir el máximo desacople de las señales neuronales obtenidas, a partir de las curvas de respuesta del sensor. Variar el parámetro ϵ, para el método de extracción de características RP, también puede aportar más información acerca del odorante que se está analizando, ya que después las características obtenidas variarán por completo.

El desarrollo de un bucle cerrado en el software de la plataforma que permita adaptar los parámetros de captura iniciales, en base a los fallos obtenidos al clasificar los datos utilizando un clasificador ya entrenado para tal propósito, para la captura de datos con un menor error y permitiendo diferenciar mejor los odorantes.

Por último, realizar experimentos con un mayor tiempo de captura para los odorantes. De este modo, además de tener más información acerca del comportamiento de cada odorante, durante el periodo de análisis, se conseguirán más datos acerca de la humedad y temperatura ambientales, pudiendo aplicar la técnica de decorrelación expuesta en (Huerta et al., 2016). Eliminando el efecto provocado por la temperatura y humedad ambientales, se podrá comprobar si los resultados que se obtengan mejoran los presentes.

Bibliografía

Adafruit (2012). adafruitTyHsensor. `https://learn.adafruit.com/dht/overview`.

Barsan, N. and Weimar, U. (2003). Understanding the fundamental principles of metal oxide based gas sensors; the example of co sensing with sno2 sensors in the presence of humidity. *Journal of Physics: Condensed Matter*, 15(20):R813.

Bouveresse, E. and Massart, D. (1996). Improvement of the piecewise direct standardisation procedure for the transfer of nir spectra for multivariate calibration. *Chemometrics and intelligent laboratory systems*, 32(2):201–213.

Buehler, M. G. and Ryan, M. A. (1997). Temperature and humidity dependence of a polymer-based gas sensor. In *Electro-Optical Technology for Remote Chemical Detection and Identification II*, volume 3082, pages 40–49. International Society for Optics and Photonics.

Cavicchi, R. E., Suehle, J. S., Kreider, K. G., Gaitan, M., and Chaparala, P. (1996). Optimized temperature-pulse sequences for the enhancement of chemically specific response patterns from micro-hotplate gas sensors. *Sensors and Actuators B: Chemical*, 33(1-3):142–146.

Cruz Gutiérrez, S. d. l. (2016). Desarrollo de una plataforma para discriminación de odorantes mediante técnicas de modulación dinámica. B.S. thesis.

Española, R. R. A. (2010). *Ortografía de la lengua española*. Espasa.

Fearn, T. (2000). On orthogonal signal correction. *Chemometrics and intelligent laboratory systems*, 50(1):47–52.

Fernandez, L., Guney, S., Gutierrez-Galvez, A., and Marco, S. (2016). Calibration transfer in temperature modulated gas sensor arrays. *Sensors and Actuators B: Chemical*, 231:276–284.

Figaro (2016a). Operating principle in mos type sensors. `http://www.figarosensor.com/technicalinfo/principle/mos-type.html`.

Figaro (2016b). TGS2600 Product information. `http://www.figarosensor.com/products/2600pdf.pdf`.

Fonollosa, J., Fernández, L., Gutiérrez-Gálvez, A., Huerta, R., and Marco, S. (2016). Calibration transfer and drift counteraction in chemical sensor arrays using direct standardization. *Sensors and Actuators B: Chemical*, 236:1044–1053.

Fonollosa, J., Fernández, L., Huerta, R., Gutiérrez-Gálvez, A., and Marco, S. (2013). Temperature optimization of metal oxide sensor arrays using mutual information. *Sensors and Actuators B: Chemical*, 187:331–339.

García Saura, C. et al. (2014). Estrategias cooperativas de detección y localización de olores con robots y narices artificiales. B.S. thesis.

Gift, N. and Jones, J. M. (2008). *Python for Unix and Linux system administration.* .ºReilly Media, Inc.".

Gosangi, R. and Gutierrez-Osuna, R. (2013). Active temperature modulation of metal-oxide sensors for quantitative analysis of gas mixtures. *Sensors and Actuators B: Chemical*, 185:201–210.

Gutierrez-Osuna, R., Gutierrez-Galvez, A., and Powar, N. (2003). Transient response analysis for temperature-modulated chemoresistors. *Sensors and Actuators B: Chemical*, 93(1):57–66.

Herrero-Carrón, F., Yáñez, D. J., de Borja Rodríguez, F., and Varona, P. (2015). An active, inverse temperature modulation strategy for single sensor odorant classification. *Sensors and Actuators B: Chemical*, 206:555–563.

Hindmarsh, J. L. and Rose, R. (1984). A model of neuronal bursting using three coupled first order differential equations. *Proc. R. Soc. Lond. B*, 221(1222):87–102.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544.

Hossein-Babaei, F. and Amini, A. (2012). A breakthrough in gas diagnosis with a temperature-modulated generic metal oxide gas sensor. *Sensors and Actuators B: Chemical*, 166:419–425.

Hossein-Babaei, F. and Amini, A. (2014). Recognition of complex odors with a single generic tin oxide gas sensor. *Sensors and Actuators B: Chemical*, 194:156–163.

Huerta, R., Mosqueiro, T., Fonollosa, J., Rulkov, N. F., and Rodriguez-Lujan, I. (2016). Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169–176.

Instrument, T. (2013). BBB Main Page. `https://beagleboard.org/black`.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572.

Izhikevich, E. M. (2007). *Dynamical systems in neuroscience*. MIT press.

Ji, W., Viscarra Rossel, R., and Shi, Z. (2015). Improved estimates of organic carbon using proximally sensed vis–nir spectra corrected by piecewise direct standardization. *European Journal of Soil Science*, 66(4):670–678.

Jing, Y.-Q., Meng, Q.-H., Qi, P.-F., Zeng, M., and Liu, Y.-J. (2016). Signal processing inspired from the olfactory bulb for electronic noses. *Measurement Science and Technology*, 28(1):015105.

Lee, A. P. and Reedy, B. J. (1999). Temperature modulation in semiconductor gas sensing. *Sensors and Actuators B: Chemical*, 60(1):35–42.

Liu, Y., Cai, W., and Shao, X. (2014). Standardization of near infrared spectra measured on multi-instrument. *Analytica chimica acta*, 836:18–23.

Macías, M. M., Agudo, J. E., Manso, A. G., Orellana, C. J. G., Velasco, H. M. G., and Caballero, R. G. (2013). A compact and low cost electronic nose for aroma detection. *Sensors*, 13(5):5528–5541.

Macías, M. M., Manso, A. G., Orellana, C. J. G., Velasco, H. M. G., Caballero, R. G., and Chamizo, J. C. P. (2012). Acetic acid detection threshold in synthetic wine samples of a portable electronic nose. *Sensors*, 13(1):208–220.

Marco, S. and Gutierrez-Galvez, A. (2012). Signal and data processing for machine olfaction and chemical sensing: A review. *IEEE Sensors Journal*, 12(11):3189–3214.

Martinelli, E., Catini, A., and Di Natale, C. (2013a). An active temperature modulation of gas sensor based on a self-adaptive strategy. In *Solid-State Sensors, Actuators and Microsystems (TRANSDUCERS & EUROSENSORS XXVII), 2013 Transducers & Eurosensors XXVII: The 17th International Conference on*, pages 2045–2048. IEEE.

Martinelli, E., Falconi, C., DAmico, A., and Di Natale, C. (2003). Feature extraction of chemical sensors in phase space. *Sensors and Actuators B: Chemical*, 95(1-3):132–139.

Martinelli, E., Magna, G., De Vito, S., Di Fuccio, R., Di Francia, G., Vergara, A., and Di Natale, C. (2013b). An adaptive classification model based on the artificial immune system for chemical sensor drift mitigation. *Sensors and Actuators B: Chemical*, 177:1017–1026.

Martinelli, E., Polese, D., Catini, A., DAmico, A., and Di Natale, C. (2012). Self-adapted temperature modulation in metal-oxide semiconductor gas sensors. *Sensors and Actuators B: Chemical*, 161(1):534–541.

Morante, J. (2013). Chemical to electrical transduction mechanisms from single metal oxide nanowire measurements: response time constant analysis. *Nanotechnology*, 24(44):444004.

Muezzinoglu, M. K., Vergara, A., Huerta, R., Rulkov, N., Rabinovich, M. I., Selverston, A., and Abarbanel, H. D. (2009). Acceleration of chemo-sensory information processing using transient features. *Sensors and Actuators B: Chemical*, 137(2):507–512.

Nakata, S., Akakabe, S., Nakasuji, M., and Yoshikawa, K. (1996). Gas sensing based on a nonlinear response: discrimination between hydrocarbons and quantification of individual components in a gas mixture. *Analytical chemistry*, 68(13):2067–2072.

Ngo, K. A., Lauque, P., and Aguir, K. (2007). High performance of a gas identification system using sensor array and temperature modulation. *Sensors and Actuators B: Chemical*, 124(1):209–216.

Ortega, Á. F. (2017). Desarrollo de un lenguaje de experimentación para una plataforma de adquisición de odorantes.

Pardo, A., Marco, S., and Samitier, J. (1998). Nonlinear inverse dynamic models of gas sensing systems based on chemical sensor arrays for quantitative measurements. *IEEE Transactions on Instrumentation and Measurement*, 47(3):644–651.

Pearce, T. C., Schiffman, S. S., Nagle, H. T., and Gardner, J. W. (2006). *Handbook of machine olfaction: electronic nose technology*. John Wiley & Sons.

Pequeño Zurro, A. (2015). Uso de una nariz electrónica ultra portátil en robots para la detección de fuentes de odorantes. B.S. thesis.

Phillips, D. (2010). *Python 3 Object Oriented Programming*. Packt Publishing Ltd.

Polese, D., Martinelli, E., Marco, S., Di Natale, C., and Gutierrez-Galvez, A. (2014). Understanding odor information segregation in the olfactory bulb by means of mitral and tufted cells. *PloS one*, 9(10):e109716.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (1996). *Numerical recipes in C*, volume 2. Cambridge university press Cambridge.

Rodriguez-Lujan, I., Bailador, G., Sanchez-Avila, C., Herrero, A., and Vidal-De-Miguel, G. (2013). Analysis of pattern recognition and dimensionality reduction techniques for odor biometrics. *Knowledge-Based Systems*, 52:279–289.

Valova, I., Lapteva, O., and Gueorguieva, N. (2007). Modeling of neurotransmitter effects in olfactory bulb. *Neural Computing and Applications*, 16(4-5):341–353.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Vázquez Rubio, T. et al. (2013). Integración de una nariz electrónica ultra-portátil en un robot modular para el control de su movimiento a través de los odorantes recibidos. B.S. thesis.

Velasco Botina, C. M. (2017). Desarrollo, estudio y análisis de una plataforma de nariz electrónica en lazo cerrado.

Vergara, A., Llobet, E., Brezmes, J., Ivanov, P., Cané, C., Gracia, I., Vilanova, X., and Correig, X. (2007). Quantitative gas mixture analysis using temperature-modulated micro-hotplate gas sensors: Selection and validation of the optimal modulating frequencies. *Sensors and Actuators B: Chemical*, 123(2):1002–1016.

Vergara, A., Llobet, E., Brezmes, J., Vilanova, X., Ivanov, P., Gràcia, I., Cané, C., and Correig, X. (2005). Optimized temperature modulation of micro-hotplate gas sensors through pseudorandom binary sequences. *IEEE Sensors Journal*, 5(6):1369–1378.

Vergara, A., Muezzinoglu, M. K., Rulkov, N., and Huerta, R. (2010). Information-theoretic optimization of chemical sensors. *Sensors and Actuators B: Chemical*, 148(1):298–306.

Vergara, A., Vembu, S., Ayhan, T., Ryan, M. A., Homer, M. L., and Huerta, R. (2012). Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166:320–329.

Wang, C., Yin, L., Zhang, L., Xiang, D., and Gao, R. (2010). Metal oxide gas sensors: sensitivity and influencing factors. *Sensors*, 10(3):2088–2106.

Wang, Y., Lysaght, M. J., and Kowalski, B. R. (1992). Improvement of multivariate calibration through instrument standardization. *Analytical Chemistry*, 64(5):562–564.

Wilson, D. M. and DeWeerth, S. P. (1995). Odor discrimination using steady-state and transient characteristics of tin-oxide sensors. *Sensors and Actuators B: Chemical*, 28(2):123–128.

Wise, B. M. and Roginski, R. T. (2015). A calibration model maintenance roadmap. *IFAC-PapersOnLine*, 48(8):260–265.

Yáñez, D. J. (2009). Estrategias bioinspiradas para la adquisición de olores en narices artificiales. *Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid*.

Yáñez, D. J., Toledano, A., Serrano, E., de Rosales, A. M. M., Rodríguez, F. B., and Varona, P. (2012). Characterization of a clinical olfactory test with an artificial nose. *Frontiers in neuroengineering*, 5.

Zbilut, J. P. and Webber Jr, C. L. (1992). Embeddings and delays as derived from quantification of recurrence plots. *Physics letters A*, 171(3-4):199–203.

A

Configuración BBB

En este anexo se explican los pasos necesarios para llevar a cabo la instalación y configuración de la plataforma. La mayor parte de este anexo ha sido obtenido del trabajo previo (Ortega, 2017), debido a que el proceso que debe realizarse para la puesta a punto de la plataforma es el mismo.

A.1. Instalación del SO

Antes de proceder con la instalación del SO en la BBB, primero es recomendable realizar una copia de seguridad de los datos que tengamos guardados, si tenemos alguno, para evitar su pérdida.

La página web oficial de la BBB, http://beagleboard.org/latest-images, tiene varios sistemas operativos donde elegir. Para este caso hemos escogido como SO Debian con su versión 7.11.

Para instalar esta imagen hay que guardarla previamente en una tarjeta SD. Para ello se puede utilizar el programa Etcher: https://etcher.io/, tal como se indica en la página oficial de la BBB: https://beagleboard.org/getting-started.

Es importante que la imagen que vayamos a grabar se tenga un formato .img y no .img.xz, ya que este último formato indica que la imagen se encuentra comprimida.

Antes de proceder con la instalación de la imagen en la BBB, hay que insertar la memoria SD en un ordenador e ir al fichero uEnv.txt. Una vez abierto este fichero, hay que ir a las lineas:

- ##enable BBB: eMMC Flasher:
- #cmdline=init=/opt/scripts/tools/eMMC/init-eMMC-flasher-v3.sh

y descomentar la última linea, borrando la almohadilla inicial, permitiendo que el sistema se instale en la BBB.

Para proceder a la instalación, descomentamos las fuentes de alimentación de la placa, insertamos la tarjeta SD en ella y mientras pulsamos el botón USER/BOOT, ver figura A.1, conectamos la BBB a la corriente.

Una vez conectada, las luces USRX, ver figura A.1, comenzarán a parpadear en orden, volcando el contenido de la tarjeta SD a la memoria eMMC de la BBB. Una vez finalizado el proceso, las luces dejarán de parpadear y el proceso habrá finalizado estando la plataforma lista para su uso. Su duración de este proceso tiene un tiempo aproximado de 30 minutos.

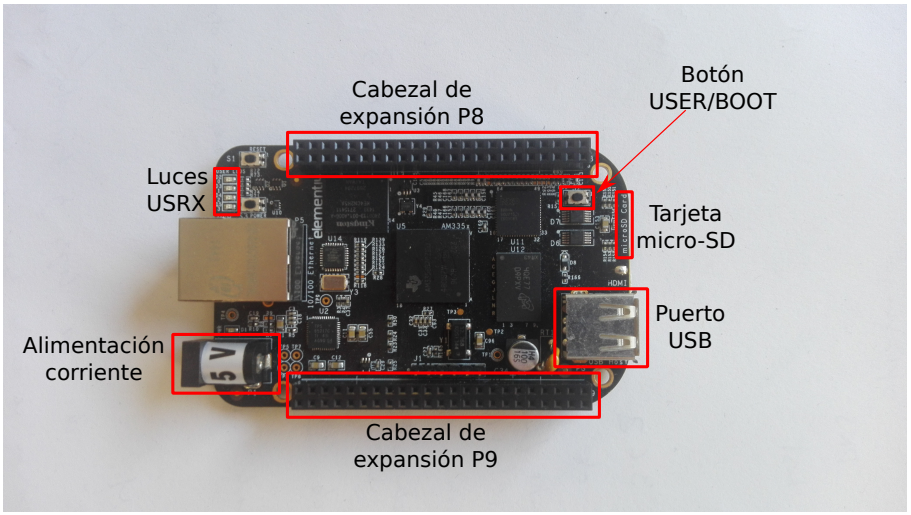


Figura A.1: Imagen de la plataforma con los diferentes elementos que la componen indicados.

A.2. Configuración BBB

Para conectarse por cable USB a la BBB, en una terminal se introduce lo siguiente: ssh root@192.168.7.2. Como se verá, no pide contraseña. Al conectarse por ssh, se ha conectado como root, que viene sin contraseña. También hay otro usuario, llamado debian. Viene por defecto. Si decidimos conectar un monitor y un teclado a la placa, veremos que arranca una interfaz gráfica. Nos logueamos y abrimos una terminal, y ya estaríamos dispuestos a continuar.

Lo siguiente es crear un usuario nuevo, o usar el usuario debian que viene por defecto, y poner contraseñas a los nuevos usuarios

Una vez logueado como super usuario, hay que poner una contraseña a este usuario. Para ello hay que introducir el siguiente comando:

- passwd

Introducimos la contraseña las veces que se requiera y con eso ya está cambiada. La contraseña tiene que ser segura, tiene que tener mayúsculas, minúsculas, caracteres numéricos y símbolos. No tiene que ser demasiado corta pero tampoco muy larga y que se pueda olvidar fácilmente.

Si se decide a crear un usuario nuevo, hay que introducir los comandos:

- adduser *new_user*

Lo siguiente que nos pedirá será una contraseña para nuestro usuario nuevo. Para ello se recomienda poner una contraseña segura, como se ha dicho anteriormente.

Nos pedirá que repitamos la contraseña, y después pedirá que introduzcamos información adicional, como nombre, apellidos, teléfono, etc. Con introducir como nombre el nombre del usuario nuevo vale. El resto puede dejarse vacío.

Si nos hemos creado un usuario nuevo, y no queremos que el usuario debian este creado, podemos borrarlo usando el comando:

- userdel -r debian

Con esto borramos el usuario debian, y su carpeta personal del sistema(bandera -r).

Puede que al intentar borrar el usuario debian, el sistema nos comunique que hay un proceso usando ese usuario. Para poder borrarlo hay que matar ese proceso(kill -9 *id proceso*) y luego volvemos a ejecutar el comando de borrado.

Si, por el contrario, queremos usar el usuario debian, y no crear uno nuevo, es muy importante cambiar la contraseña y no dejar la que viene por defecto. Para ello hay que introducir el comando: pwd cuando nos hayamos logueado con este usuario. Nos pedirá una contraseña, en este caso, como en el anterior, hay que poner una contraseña segura.

Importante Si nos hemos logueado en la BBB con el comando anterior(ssh root@192.168.7.2), estamos en el usuario root. Para cambiar de usuario, se puede hacer de dos formas:

- Saliendo de esa sesión, usando el comando: exit, y entrando con el comando: ssh debian@192.168.7.2. En este caso, este usuario si que pide una contraseña, que es: tmppwd. La introducimos y ya estamos en el usuario debian.
- Ponemos el comando: su debian. Este comando lo que hace es cambiar entre usuarios. Le indicamos que se cambie al usuario debian. Nos pedirá la contraseña; la introducimos y ya estamos en el usuario debian.

Para que nuestro usuario, ya sea uno nuevo o el usuario debian por defecto pueda realizar acciones de super usuario, hay que añadirle al grupo sudo. Para ello se pueden usar dos comandos distintos:

- adduser *user* sudo
- usermod -a -G sudo *user*

Importante Para añadir un usuario al grupo sudo, seguramente pida permisos de super usuario. Para ello tenemos que estar logueados como sudo. Puede que haya que reiniciar la BBB para que el cambio surga efecto.

Para que la BBB tenga acceso a la red y poner conectarse en remoto hay que configurar la interfaz de red y los DNS.

Para configurar la interfaz de red, hay que cambiar el fichero /etc/network/interfaces, que es el encargado de la interfaz de red. Para cambiar este fichero hay que hacerlo como super usuario.

Para editar el fichero ponemos uno de los siguiente comandos:

- sudo vi /etc/network/interfaces
- sudo nano /etc/network/interfaces

Se abrirá el fichero y podremos observar su contenido, vamos al final del fichero y añadimos las siguientes lineas, guardamos y salimos:

- auto eth0

- iface eth0 inet static
- address *IP asignada*
- netmask *IP de la máscara*
- gateway *IP del gateway*

Para los servidores DNS hay dos formas de hacerlo dependiendo de la versión de sistema operativo que tengamos en la BBB. Primero vamos al fichero: /etc/resolv.conf; si en ese fichero hay un mensaje que nos dice que no modifiquemos el contenido porque se borrará, no cambiamos nada; si no nos aparece ningún mensaje, modificamos el fichero con lo siguiente:

- sudo vi /etc/resolv.conf
- sudo nano /etc/resolv.conf

Una vez dentro borramos todo lo que hay en el fichero, y ponemos lo siguiente:

- nameserver: *ip del DNS1 ip del DNS2 ip del DNS3*, ...

Ponemos todas las IPs seguidas por comas.

Si en el fichero nos ponía que no lo modificásemos, las DNS se configuran en el fichero /etc/network/interfaces. Lo abrimos y al final añadimos lo siguiente:

- dns-nameservers *ip del DNS1*,*ip del DNS2*, ...

Para cambiar el nombre al host por el que nos han proporcionado hay que cambiar dos ficheros:

- /etc/hosts
- /etc/hostname

Para el primero, lo abrimos con uno de los dos comandos:

- sudo vi /etc/hosts
- sudo nano /etc/hosts

De este fichero solo interesan las dos primeras lineas:

- 127.0.0.1 localhost
- 127.0.1.1 **beaglebone**.localdomain **beaglebone**

Estas dos primeras lineas empiezan con una IP, ambas. Luego viene información sobre como el sistema tiene configurado los distintos hosts. En este caso, solo nos interesan las palabras en negrita. Hay que sustituirlas por el nombre del host que hayan asignado a la BBB.

Del segundo fichero(/etc/hostname), hay que borrar su contenido(contiene el nombre del host), y escribir el nombre que le hayan aasignado a la BBB.

Con esto ya se habría configurado la interfaz de red y los DNS. Solo falta reiniciar la BBB para que los cambios se apliquen:

- sudo reboot

o usar el siguiente comando para reiniciar la interfaz de red:

- sudo service networking restart

Para el servidor ssh, es recomendable cambiar ciertos parámetros que vienen configurados por defecto. Para realizar estos cambios hay que cambiar el fichero:

- /etc/ssh/sshd_config

Accedemos al fichero con vi o nano, y buscamos y cambiamos los siguientes parámetros:

- Port X(donde X es cualquier puerto que queramos poner). El puerto estándar es el 22, es muy recomendable cambiar este puerto a otro.
- PermitRootLogin no. Por defecto está a yes, pero es recomendable ponerlo a no y evitar conectarse al usuario root desde ssh
- MaxAuthTries X. El número máximo de intentos antes de que el sistema desautorice esa conexión.
- AllowUsers X. Donde X son los usuarios del sistema que vamos a permitir conectarse por ssh, separados por espacios

Si además queremos usar un certificado público-privado como clave para conectarse, en vez de tener que introducir la contraseña, hay que cambiar o añadir algún campo más:

- RSAAuthentication yes
- PubkeyAuthetication yes
- PasswordAuthentication no

Con estos cambios, ahora solo podría accederse usando un certificado público-privado. **Importante** Si configuramos la BBB para que solo se pueda conectar por certificado es muy importante poner la clave pública del certificado en la BBB en la siguiente ruta, antes de reiniciar el servidor ssh, o la BBB:

- /home/*username*/.ssh/authorized_keys

Si la carpeta no existe, creamos una carpeta nueva con el comando:

- mkdir .ssh

Al empezar por . la carpeta será una carpeta oculta. Dentro de la carpeta, copiamos la clave pública. Si el fichero no existe, lo creamos con el comando:

- touch authorized_keys

Para introducir el contenido de la clave pública en el fichero, si la tenemos en la BBB, usamos el comando:

- cat *clave_publica* » authorized_keys

La BBB, por defecto trae la hora de Londres. Para cambiar al huso horario deseado, primero tenemos que ir al siguiente directorio:

- /usr/share/zoneinfo/Europe/

Usamos el comando ls, y vemos todos las ciudades que hay en ese directorio. Escogemos la ciudad de la que queramos escoger la hora. En nuestro caso escogemos Madrid. Una vez escogida la ciudad deseada, hay que borrar el fichero:

- /etc/localtime

Seguramente requiera permisos de super usuario. Este fichero es el que regula la hora del sistema. Una vez borrado, hay que crear un enlace simbólico a la ciudad que hayamos elegido. Un enlace simbólico no es más que un fichero que hace referencia a otro fichero, situado en otro lugar del sistema. Para crear el enlace ponemos:

- ln -s /usr/share/zoneinfo/Europe/Madrid /etc/localtime

Tras esto simplemente reiniciamos la BBB:

- sudo reboot

Es recomendable utilizar una tarjeta SD en la BBB para poder guardar en ella todos los experimentos que hagamos, teniendo así una copia de seguridad. Utilizando la tarjeta SD vamos a añadir memoria SWAP a la BBB para poder realizar la instalación de los paquetes Python, ya que sino no es posible realizar la instalación de estos. Es importante recordar que el uso de una tarjeta de memoria SD como memoria SWAP puede acortar su periodo de vida, debido a su continuo uso.

Para ello, lo primero que hay que hacer es introducir la tarjeta de memoria en la BBB, estando esta apagada. Después es necesario montarla, para poder acceder a los archivos, si es que no se ha montado directamente. Para montar la tarjeta de memoria y poder acceder a su contenido, ponemos el comando:

- fdisk -l

Este comando nos muestra los dispositivos de almacenamiento que se encuentran disponibles en el sistema. La tarjeta de memoria insertada, seguramente, tendrá el identificador: /dev/mmcblk1p1, aunque conviene asegurarse. Para ello, antes de insertar la tarjeta SD, introducir este comando, ver la salida y comparar esta con la salida obtenida después de introducir la tarjeta de memoria. El significado de mmcblk1p1 es el siguiente:

- **mmc**: **m**ulti**m**edia **c**ard.
- **blk1**: **bl**oc**k** device 1.
- **p1**: **p**artition 1.

Para montar la tarjeta y poder acceder a su contenido se utiliza el comando: mount. Antes de ello, creamos una carpeta en la localización donde queramos que se monte la tarjeta, en nuestro caso se creó en: */media/sdcard/*, aunque puede ser en cualquier otra localización. Una vez que tengamos la carpeta donde queremos montar la tarjeta, introducimos el comando:

- mount -v /dev/mmcblk1p1/ /media/sdcard/.

Con esto ya tendremos acceso a la tarjeta de memoria. Antes de automatizar el proceso de montaje de la tarjeta durante el arranque, vamos a crear una partición de memoria SWAP para la instalación de las diferentes librerías de Python. El tamaño escogido para esta particiçon es de 2GB. Para reservar espacio en la tarjeta de memoria utilizamos el comando fallocate:

- sudo fallocate -l 2GB /media/sdcard/swapfile1

Con este comando reservamos 2GB de memoria física en la localización /media/sdcard/ en un fichero llamado swapfile1. Si queremos comprobar que se ha reservado bien, podemos escribir el siguiente comando:

- ls -la /media/sdcard/swapfile1

Una vez que hayamos comprobado que se ha creado el fichero correctamente, es de extrema importancia cambiar tanto los permisos como el dueño del fichero. Esto es importante porque sino cualquier usuario podría leer el contenido del fichero de SWAP, creando un agujero en la seguridad del sistema. Para cambiar el dueño del fichero al superusuario y cambiar los permisos del fichero introducimos los siguientes comandos:

- sudo chown -R sudo /media/sdcard/swapfile1
- sudo chmod 600 /media/sdcard/swapfile1

Con estos comandos hemos cambiado el usuario del fichero de SWAP al superusuario, y los permisos para que solo el superusuario pueda leer y escribir en ese fichero.

Por último solo falta automatizar la carga de la tarjeta SD y del fichero de SWAP. Para ello hay que añadir dos lineas al fichero fstab, que define particiones para ser montadas en el sistema. Para añadirlos, en supersusario, añadimos al final del archivo las siguientes lineas:

- /dev/mmcblk0p1 /media/sdcard auto defaults,nofail 0 0
- /media/sdcard/swapfile1 none swap sw 0 0

La primera linea automatiza el montaje de la tarjeta de memoria SD, utilizando el flag auto. La bandera defaults agrupa un conjunto de banderas para el arranque por defecto. La bandera nofail es de gran importancia, ya que si el sistema no encuentra la tarjeta SD arrancará con normalidad, en caso contrario lanzará un fallo durante el arranque. Con la segunda línea automatizamos la carga de memoria swap que se encuentra en la tarjeta SD. Es importante el orden de carga, ya que si intentamos cargar la memoria SWAP antes que la tarjeta SD, se producirá un fallo, debido a que el archivo de memoria SWAP se encuentra contenido en la memoria SD. Todas las opciones para la automatización de las particiones se encuentran en: https://wiki.archlinux.org/index.php/fstab.

Con esto ya estaría la BBB lista para funcionar.

A.3. Instalación de paquetes Python

Antes de introducir cualquier comando de ejecución, introducimos el siguiente comando para actualizar los paquetes disponibles:

- sudo apt-get update

Después de actualizar los paquetes disponibles, actualizamos el sistema mediante el comando:

- sudo apt-get upgrade

Instalamos las dependecias necesarias para la instalación de Python3.

- sudo apt-get install build-essential -y

Una vez realizados estos pasos, procedemos con la instalación de las librerías necesarias. Para instalar la versión de Python requerida, es necesario descargarla de la página oficial: https://www.python.org/, ya que la versión 3 de Python no se encuentra en los repositorios oficiales para esta versión de Debian.

Una vez descargado el fichero comprimido, se descomprime y se va a la carpeta donde se ha descomprimido. En ella ponemos los siguientes comandos en orden:

- ./configure –prefix=/usr/
- make
- make install

Mediante estos comandos el sistema instalará esta versión de Python. Lo siguiente es instalar las librerías necesarias para poder utilizar el software. Para ello primero actualizamos las librerias python3-dev y python3-setuptools. Para ello nos ayudamos del sistema de gestión de paquetes: pip, que se ha instalado con la versión de Python 3. Los comandos necesarios para realizar la actualización son:

- sudo pip3 install –upgrade dev setuptools

Antes de proceder a la instalación del resto de paquetes de Python, es necesario instalar dos librerías de manejo de matrices en C, las cuales son utilizadas por Python mediante una capa de abstracción. Estas librerías: BLAS y LAPACK, permiten un manejo óptimo de matrices en C, habiendo sido optimizadas al máximo y siendo utilizadas en la mayor parte de las librerías de cálculo numérico de Python. Para instalarlas hay que poner los siguientes comandos:

- sudo apt-get install libblas-dev libblas-doc liblapack-dev liblapack-doc

Una vez realizada la instalación y actualización de estos paquetes, procedemos a instalar varias librerías necesarias para la utilización del software de la plataforma. La instalación de estas librerías llevará un tiempo considerable, por lo que es mejor ponerlo mientras hacemos una tarea paralela. El comando de instalación es el siguiente:

- sudo pip3 install cython numpy scipy pgen

Por último, solo falta instalar las librerías que realizan el control de los pines de la placa y permiten leer de los sensores que recogen datos de temperatura y humedad. Para instalarlas, las descargamos de la plataforma github:

I/O	Nombre	Num Pines	Rango Voltaje	Corriente max	Tiempo Muestreo
Analog (I)	AIN_X	7	0 - 1,8 V	$2\mu A$	$125ns$
Digital (I)	GPIO_X	66	0 o 3.3 V	4 - 6(mA)	140(ns)
Digital (O)	GPIO_X	66	0 o 3,3 V	4 - 6(mA)	95 - 105(ns)

Tabla A.1: Nomenclatura de los pines de la BBB y sus limitaciones físicas

- git clone https://github.com/adafruit/adafruit-beaglebone-io-python.git
- git clone https://github.com/adafruit/Adafruit_Python_DHT.git

Para el primer repositorio es de gran importancia que descarguemos la versión 1.0.3, ya que para otras versiones se producian fallos al ejecutar el algoritmo de captura por modulación en frecuencia. Una vez que hayamos descargado ambos ficheros, los descomprimimos y en la carpeta donde los hayamos descomprimido y ejecutamos el comando:

- sudo python3 setup.py install

Una vez instalados estos dos paquetes, ya habremos finalizado por completo la instalación de los diferentes paquetes en la plataforma.

A.4. Cuidado y manejo de la BBB

Para realizar cualquier acción con la BBB, antes de tocarla, hay que descargarse de electricidad estática. Para ello basta con tocar cualquier objeto metálico o con tener una pulsera anti-electricidad estática.

Si la BBB tiene un circuito montado y se quiere manipular, es muy importante apagar la BBB, y desconectar los cables de corriente de la BBB, ya que se podría producir un cortocircuito accidentalmente.

Al montar cualquier circuito, es recomendable asegurarse concienzudamente de que todos los pines están en la posición correcta, ya que es fácil confundirse y estropear la BBB.

Cuando se haya terminado de cambiar todo, volvemos a reconectar los cables de alimentación y podemos volver a usar la BBB.

A.4.1. Pines de la BBB

La BBB tiene, en total, 92 pines de expansión, distribuidos en dos cabezales de expansión. Estos pines pueden ser de distintos tipos. Los distintos tipos usados en este proyecto son:

- PWM o Pulse With Modulation. Estos pines son una salida.
- ADC o Analogic to Digital Converter. Estos pines son una entrada.
- GPIO o General Purpose Input/Output. Estos pueden ser tanto una salida como una entrada.

Estos pines tienen un conjunto de limitaciones, que hay que respetar, ya que sino la BBB puede quemarse. Las limitaciones de estos pines se pueden ver en la tabla A.1.

Si se superan estos rangos, la BBB se estropeará. Si se monta un circuito en la BBB, será necesario poner algún circuito de protección ya que si se superan estos límites se estropearía la placa.

B

Manual usuario

B.1. Manual Usuario

En este anexo se muestran los pasos que hay que seguir para poder correr los experimentos de captura.

Para poner experimentos de captura en la plataforma se pueden utilizar dos formas:

- Mediante el uso de la interfaz gráfica.
- Mediante el uso de scripts y la linea de comandos

Si decidimos utilizar la interfaz gráfica, hay que arrancarla utilizando el comando: *python3 GPyHuele*, en la carpeta donde se encuentre el programa. En ella, hay que rellenar los parámetros de configuración del experimento y de la plataforma. Una vez que se han especificado estos parámetros, tanto para la configuración del experimento como de la plataforma, se puede guardar esta configuración mediante el uso del botón save, ver figura B.1. Nos saltará una ventana para podremos escoger un nombre y localización para el ficheros. También podremos cargar una configuración mediante el botón load, ver figura B.1. Para configurar la conexión ssh del experimento, utilizamos el botón connect para acceder al menú donde se introducen los elementos relacionados con la conexión. Para ejecutar el experimento, pulsamos el botón start enviando los ficheros con los datos de configuración a la plataforma de forma remota y lanzando el comando de ejecución para iniciar el proceso de captura.

Si decidimos utilizar la linea de comandos, hay que escribir en dos ficheros los parámetros de configuración del experimento en uno y los parámetros de configuración de la plataforma en otro. Una vez que se han creado los ficheros, se pone en la linea de comandos en la plataforma el comando: *sudo python3 PyHuele.py fichero_conf_experimento fichero_conf_plataforma.*

Para ver el significado de las palabras clave que debe de utilizarse en la configuración del experimento, se recomienda leer el trabajo (Ortega, 2017), donde se detallan todos los aspectos. Para el significado y uso de la sintaxis, se recomienda leer el trabajo (Ortega, 2017) y la sección 3 de este trabajo. Por último, para realizar la configuración de la plataforma se recomienda leer la sección 3.

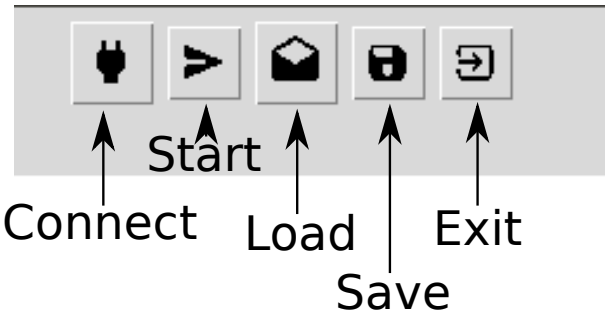


Figura B.1: Botones de la interfaz gráfica y su función.

A continuación se muestra un ejemplo de uso de la interfaz, para la captura de odorantes, utilizando los parámetros expuestos en la sección 5.2:

1. Tras arrancar el software, se mostrará la pestaña de configuración del experimento. Simplemente hay que rellenar las opciones según el experimento que se desee lanzar, como se muestra en la imagen de la izquierda en la figura B.1.

2. Para configurar las características de la plataforma, hay que seleccionar la pestaña correspondiente. Tras eso se rellenan las opciones de configuración como más se desee, como se muestra en la imagen central de la figura B.1.

3. Si en algún momento se desea guardar, o cargar, las opciones introducidas en un fichero, solamente hay que situarse en la pestaña que se desee y hacer uso de los botones save o load B.1. Tras seleccionar la opción deseada, se abrirá una ventana donde se podrá seleccionar la carpeta y nombre del fichero.

4. Para poder realizar la conexión con la plataforma, es necesario utilizar el botón: connect B.1. A continuación se abrirá un pop-up donde se deben introducir los parámetros de la conexión, tal y como se muestra en la imagen de la derecha en la figura B.1.

5. Para iniciar el experimento solamente hay que seleccionar la opción start en el menú superior B.1.

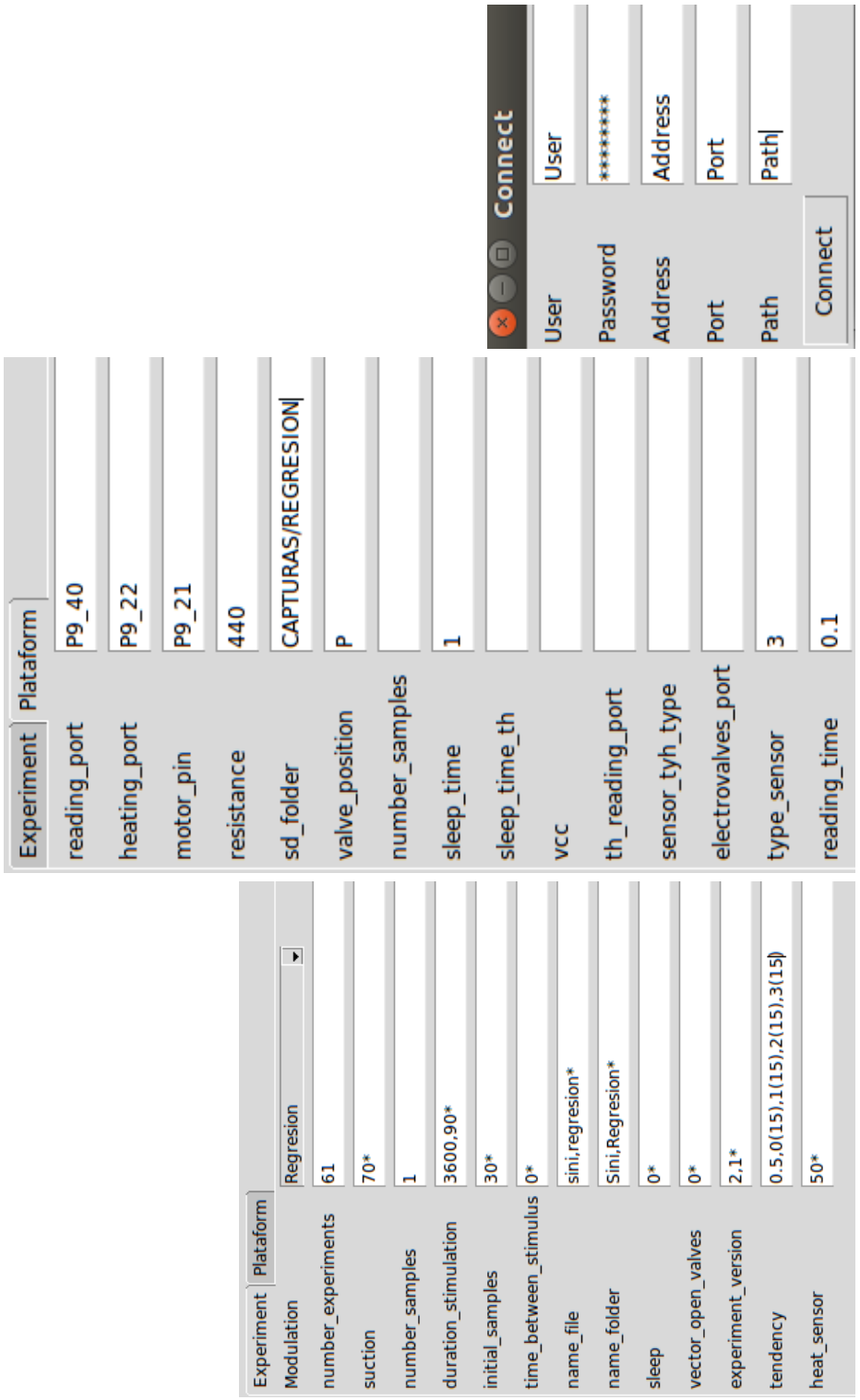


Figura B.2: Imágenes que muestran como configurar un experimento en la plataforma.

C

Codigos

En este anexo se muestra, a modo de ejemplo del trabajo de programación realizado, el código principal del módulo de captura del programa PyHuele, donde se exponen los diferentes algoritmos de experimentación. El resto del código puede encontrarse en el Github: `https://github.com/TheBarto/EN/tree/master`.

```
# -*- encoding: utf-8 -*- # Especificamos que nuestro archivo .py estÃą
    codificado en UTF-8
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.ADC as ADC
import Adafruit_DHT as DHT
import time
import os
import sys
import tratamiento_cadenas as tc
from threading import Thread,currentThread,Event
from datetime import datetime, date
from scipy import stats
import queue as qu
import numpy as np
import math
import pickle
import sys
from sklearn import svm

class Modulacion(object):

        motorPin = 'P9_21'
        NM = 10 #Numero lecturas ADC
        T = 0.1 #Las NM lecturas se hacen en T segundos
        tsub = T/NM; #Subdivisiones de tiempo para las lecturas ADC
        SLEEP, SLEEP_tyh = 1,59
        Vcc = 5
        sensorTemp22 = DHT.AM2302
        Temp22 = 'P8_11'
        odorantes = {1:'Metanol',2:'Etanol',3:'Butanol',4:'Aire'}
        TyH_sensor = {1:DHT.DHT11,2:DHT.DHT22,3:DHT.AM2302}
        electrovalvulas = ['P8_10','P8_12','P8_14','P8_16']

        def __init__(self,config_experiment,config_platform,strings,**kwargs):
```

```
                super().__init__(**kwargs) #Comprobar que no falle por esto
36              Modulacion.daemonizar()

38              self.succiones = config_experiment[tc.SSUCCION]
                self.switchs = config_experiment[tc.SSWESTIM]
40              self.samples_ini = config_experiment[tc.SSINICIO]
                self.cte = config_experiment[tc.SSBESTIM]
42              self.name_files = config_experiment[tc.SNFILE]
                self.name_folders = config_experiment[tc.SNFOLDER]
44              self.vecs_open_valves = config_experiment[tc.SVECOPVAL]
                self.vecs_anal_odort = config_experiment[tc.SVECOPODR]
46              self.sleeps = config_experiment[tc.SSLEEP]
                self.time_mark = datetime.now()
48              self.f,self.g,self.h,self.thread = None,None,None,None
                Modulacion.motorPin = config_platform[tc.SMTPORT] if
    config_platform[tc.SMTPORT] != '' else 'P9_21'
50              Modulacion.NM = int(config_platform[tc.SNMUETS]) if
    config_platform[tc.SNMUETS] != '' else 10
                Modulacion.T = float(config_platform[tc.SRDTIME]) if
    config_platform[tc.SRDTIME] != '' else 0.1
52              Modulacion.SLEEP = int(config_platform[tc.SSLEEPM]) if
    config_platform[tc.SSLEEPM] != '' else 1
                Modulacion.SLEEP_tyh = int(config_platform[tc.SSLEEPTYH]) if
    config_platform[tc.SSLEEPTYH] != '' else 59
54              Modulacion.Vcc = int(config_platform[tc.SVCC]) if config_platform
    [tc.SVCC] != '' else 5
                Modulacion.Temp22 = config_platform[tc.STEMPPORT] if
    config_platform[tc.STEMPPORT] != '' else 'P8_11'
56              Modulacion.electrovalvulas = config_platform[tc.SELECPORTS] if
    config_platform[tc.SELECPORTS] != [''] else ['P8_10','P8_12','P8_14','P8_16']
                Modulacion.sensorTemp22 = Modulacion.TyH_sensor[config_platform[
    tc.SSENTYPE]]
58              Modulacion.tsub = Modulacion.T / Modulacion.NM
                Modulacion.electrpos = config_platform[tc.SVALPOS]
60              Modulacion.capturas_iniciales = [4] if Modulacion.electrpos != 'R
    ' else [1,2,3]
                Modulacion.queue = qu.Queue() # Creo la cola de mensaje global
62              Modulacion.event = Event() # Evento para sincronizar los hilos de
     captura y de escritura
                self.activar_GPIO_valvulas() # Puede ser un foco de problemas,
    puede que haya que cambiarlo
64              self.write_thread = Thread(target=self.hilo_escritura_datos,args
    =(strings,)) # Creo el hilo de escritura global
                self.write_thread.start()
66
                try:
68                      pid = os.fork()
                        if pid > 0:
70                              sys.exit(0)
                except OSError as e:
72                      sys.sderr.write("fork #1 failed: (%d) %s\n" %(e.eerno, e
    .streerror))
                        sys.exit(1)
74
                #Separar entorno padre
76              #os.chdir("/")
                os.umask(0)
78              os.setsid()
                #Realizar segundo fork
80
                try:
82                      pid = os.fork()
                        if pid > 0:
```

```
84                                  sys.exit(0)
                except OSError as e:
86                          sys.sderr.write("fork #2 failed: (%d) %s\n" %(e.eerno, e
    .streerror))
                            sys.exit(1)
88
                for f in sys.stdout, sys.stderr,sys.stdin: f.flush()
90              si = open("salida_in.txt", 'r')
                so = open("salida_out.txt", 'w')
92              se = open("salida_err.txt", 'w')
                os.dup2(si.fileno(), sys.stdin.fileno())
94              os.dup2(so.fileno(), sys.stdout.fileno())
                os.dup2(se.fileno(), sys.stderr.fileno())
96
                pid = str(os.getpid())
98              open('file_daemon','w+').write("%s\n" %pid)

100         def activar_GPIO_valvulas(self):
                for elec in Modulacion.electrovalvulas:
102                         GPIO.setup(elec,GPIO.OUT)

104         def create_files(self,nameFile,path,folder,samplesinicio):
                """
106             Crea los ficheros para escribir
                Crea los ficheros necesarios para guardar todos los datos de las
108             experimentaciones.
                Parametros:
110             nameFile -- nombre que va a usarse para crear los ficheros que
    contendran los datos y
                el resto de la informacion
112             path -- ruta donde se va a guardar el fichero
                folder -- nombre de la carpeta donde va a guardarse el archivo
114
                Retorno:
116             devolver -- array con los gases que van a entrar al sensor
                """
118             datenow = datetime.now()

120             fileString = nameFile+"_txt_" +datenow.strftime('%Y-%m-%d_%H-%M-%
    S')+".txt"
                fileString_data = nameFile+"_dat1_"+datenow.strftime('%Y-%m-%d_%H
    -%M-%S')+".dat"
122             fileString_tyh = "TyH"+nameFile+"_data_"+datenow.strftime('%Y-%m
    -%d_%H-%M-%S')+".data"

124             ruta1 = tc.obtener_ruta(path,self.time_mark,folder,"dat") #Ruta
    de salida
                if not os.path.exists(ruta1): os.makedirs(ruta1,mode=0o755)     #
    Si la ruta no existe, se crea
126             ruta2 = tc.obtener_ruta(path,self.time_mark,folder,"txt") #Ruta
    de salida
                if not os.path.exists(ruta2): os.makedirs(ruta2,mode=0o755)     #
    Si la ruta no existe, se crea
128             ruta3 = tc.obtener_ruta(path,self.time_mark,folder,"TyH") #Ruta
    de salida
                if not os.path.exists(ruta3): os.makedirs(ruta3,mode=0o755)     #
    Si la ruta no existe, se crea
130
                return ruta2.strip() + str(fileString),ruta1.strip() + str(
    fileString_data),ruta3.strip() + str(fileString_tyh)
132
        def open_files(self,ruta_fichero,ruta_fichero_data,ruta_fichero_tyh):
134             return os.fdopen(os.open(ruta_fichero, os.O_WRONLY | os.O_CREAT |
```

```
    os.O_TRUNC, 0o644), 'w'),os.fdopen(os.open(ruta_fichero_data, os.O_WRONLY |
    os.O_CREAT | os.O_TRUNC, 0o644), 'w'),os.fdopen(os.open(ruta_fichero_tyh, os.
    O_WRONLY | os.O_CREAT | os.O_TRUNC, 0o644), 'w')

136     def file_TGS2600(self,ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_anal_odor,succion,heat2600,tiempo,switch,tespera,samplesinicio):
            self.f.write('Configuracion de la plataforma\n')
138         self.f.write('Pin del motor: '+str(Modulacion.motorPin)+'\n')
            self.f.write('Numero de capturas ADC para una muestra: '+str(
    Modulacion.NM)+'\n')
140         self.f.write('Tiempo en que se realizan las NM capturas de una
    muestra: '+str(Modulacion.T)+'\n')
            self.f.write('Tiempo limite de captura de una muestra: '+str(
    Modulacion.SLEEP)+'\n')
142         self.f.write('Tiempo limite de captura de una muestra de TyH: '+
    str(Modulacion.SLEEP_tyh)+'\n')
            self.f.write('Valor de VCC del circuito: '+str(Modulacion.Vcc)+'
    \n')
144         self.f.write('El puerto de lectura de la humedad y temperatura
    es el: '+str(Modulacion.Temp22)+'\n')
            self.f.write('Los puertos de las electrovalvulas son: '+str(
    Modulacion.electrovalvulas)+'\n')
146         self.f.write('Posicion de las electrovalvulas: '+Modulacion.
    electrpos+'\n')
            self.f.write('Sensor TGS2600\n')
148         self.f.write('Fecha y hora de inicio: ' + str(time.strftime("%a %
    d %b %Y - %H:%M:%S", time.localtime())) + '\n')
            self.f.write('Ruta del fichero: ' + str(ruta) + '\n')
150         self.f.write('Nombre del fichero: ' + str(nameFile) + '\n')
            self.f.write('Nombre del fichero de datos 1: ' + str(
    nameFile_data1) + '\n')
152         self.f.write('Nombre del fichero de TyH: ' + str(nameFile_tyh) +
     '\n')
            self.f.write('Electrovalvulas (1-METANOL, 2-ETANOL, 3-BUTANOL,
    4-AIRE ) \n')
154         self.f.write('Conmutacion entre electrovalvulas: ' +str(
    vec_anal_odor)+ '\n')
            self.f.write('Succion motor (30-100)   >>> '+ str(succion)+ '%\n
    ')
156         self.f.write('Temperatura TGS2600 (1-100) >>> '+str(heat2600)+ '
    \n') #+' Pin PWM : ' +str(heatPin2600)
            self.f.write('Duracion del experimento: ' +str(tiempo)+ '
    minutos o muestras\n')
158         self.f.write('Se ha dejado un tiempo entre captura de gases de:
    '+str(tespera)+ ' segundos\n')
            self.f.write('El experimento tiene: ' +str(samplesinicio)+ '
    muestras iniciales\n')
160         self.f.write('El tiempo de switch o conmutacion de las
    electrovalvulas es de: '+str(switch)+'\n')

162     def cerrar_electrovalvulas(self):
            for electrovalvula in Modulacion.electrovalvulas:
164             GPIO.output(electrovalvula, GPIO.LOW)

166     def abrir_electrovalvulas(self,electrovalvulas):
            for number_elect in electrovalvulas:
168             GPIO.output(Modulacion.electrovalvulas[number_elect-1],
    GPIO.HIGH)

170     def measure_tyh(self):

172         self.thread = currentThread()
            humidity,temperature = 0,0
```

```
174
                #Lectura humedad y temperatura
176             while getattr(self.thread,"do_run",True):
                        tick_HT = time.time()
178                     humidity, temperature = DHT.read_retry(Modulacion.
    sensorTemp22,Modulacion.Temp22,30,1,None)#read_retry(sensorTemp22,Temp22)
                        t_HT = time.time()-tick_HT
180                     instante = datetime.now()

182                     #Cuanto duerme en funcion de lo que tarde en H y T
                        if t_HT > Modulacion.SLEEP_tyh:
184                             print ("Tiempo medicion H y T > SLEEP:", t_HT)
                        else:
186                             print ("Tiempo medicion H y T:", t_HT)
                                print ('\n Sensor DHT22: ' + str(Modulacion.
    sensorTemp22))
188
                                if humidity is not None and temperature is not
    None:
190                                     print ('>>> '+str(instante)+'Temp = ' +
    str(temperature) + 'Humidity = ' + str(humidity) + '\n')
                                else :
192                                     print ('Failed to get reading, Try again!
    ')

194                             #h = open(ruta_fichero_tyh, "a")
                                wline_h = str(instante) + ' ' + str(temperature)
    + ' ' + str(humidity) + '\n'
196                             self.h.writelines(wline_h)
                                self.h.flush()
198
                        time.sleep(Modulacion.SLEEP_tyh-(time.time()-tick_HT))
200
                return 0
202
        def cierre(self,heatPin):
204
                Modulacion.queue.put(0)
206             Modulacion.event.wait() # Me sincronizo con el hilo de escritura
    para asegurarme de que ha tenido tiempo para escribir todo
                self.cerrar_electrovalvulas()
208             self.thread.do_run = False
                self.thread.join()
210             self.motor_stop()
                PWM.stop(heatPin)
212             PWM.cleanup()
                fecha_fin = datetime.now()
214             print ('Experimento terminado: '+str(fecha_fin))
                self.f.write('\nEXPERIMENTO FINALIZADO CON EXITO\n')
216             self.f.write('Fecha y hora de fin de experimentacion:' +str(
    fecha_fin))
                self.f.flush()
218             self.f.close()
                self.g.close()
220             self.h.close()
                return
222
        def captura_odorante(self):
224             raise Exception("subclasses must override captura_odorante()!")

226     def valor_sensor(self):
                raise Exception("subclasses must override valor_sensor()!")
228
```

```
        def imprimir_cabecera(self):
230             raise Exception("subclasses must override imprimir_cabecera()!")

232     def motor_start(self,succion):
            PWM.start(Modulacion.motorPin,succion)
234
        def motor_stop(self):
236             PWM.stop(Modulacion.motorPin)

238     def hilo_escritura_datos(self,strings):

240             while(1):
                    values = Modulacion.queue.get()
242                 if values == -1:
                        Modulacion.event.set()
244                     return 0
                    if values == 0:
246                     Modulacion.event.set()
                        continue
248
                    str2 = values.pop(0)+': '
250                 count = values.pop(0)
                    gases = values.pop()
252                 str1 = str(count)+' '
                    str2 += str(count)+' '
254
                    for value,string in zip(values,strings):
256                         str1+=str(value)+' '
                            str2+=string+' '+str(value)+' '
258                 str2+='Gas(es) captados e identificadores: '

260                 for gas in gases:
                            str1+=str(gas)+' '+Modulacion.odorantes[gas]
262                         str2+=str(gas)+' '+Modulacion.odorantes[gas]

264                 self.g.write(str1+'\n')
                    self.g.flush()
266                 self.f.write(str2+'\n')
                    self.f.flush()
268             return

270     def crear_hilo_TyH(self):
                self.thread = Thread(target=self.measure_tyh,args=())
272             self.thread.do_run = True
                self.thread.start()
274
        #Puro.path --> tiene que ser un path de cada uno
276     def apertura_escritura_ficheros(self,path,sw,samplesinicio,ct,nfile,
    nfolder,vsaodrs,arg_extra=None):
                ruta_fichero,ruta_fichero_data,ruta_fichero_tyh = Modulacion.
    create_files(self,nfile,path,nfolder,samplesinicio)
278             self.f,self.g,self.h = Modulacion.open_files(self,ruta_fichero,
    ruta_fichero_data,ruta_fichero_tyh)

280             print ('Ruta Fichero: ' + ruta_fichero)
                print ('Ruta Fichero de datos: '+ ruta_fichero_data)
282             print ('Ruta Fichero de datos TyH: '+ ruta_fichero_tyh)

284             return

286     def inicializar_hilos_puertos(self,suc):
                Modulacion.crear_hilo_TyH(self)
288             Modulacion.motor_start(self,suc)
```

```
290         def capturas_muestras_iniciales(self,samplesinicio):
                print ('\n\nComienza la adquisicion. Muestras inciales: ' +str(
    samplesinicio)+ '\n\n')
292             self.f.write('\n\nComienza la adquisicion. Muestras inciales: ' +
    str(samplesinicio)+ '\n\n')
                self.f.flush()
294             Modulacion.cerrar_electrovalvulas(self)

296         def captura_muestras(self):
                print ('\n\nComienza la experimentacion\n')
298             self.f.write('\n\nComienza la experimentacion\n')
                self.f.flush()
300             Modulacion.cerrar_electrovalvulas(self)

302         def captura_datos(self,*args):

304             for arg in zip(self.succiones,self.switchs,self.samples_ini,self.
    cte,self.name_files,
                        self.name_folders,self.vecs_open_valves,self.
    vecs_anal_odort,self.sleeps,*args):
306                 self.apertura_escritura_ficheros(arg[1],arg[2],arg[3],arg
    [4],arg[5],arg[7],arg[9:])
                    self.inicializar_hilos_puertos(arg[0],arg[1],arg[2],arg
    [3],arg[4],arg[5],arg[6],arg[7],arg[9:])
308                 self.capturas_muestras_iniciales(arg[2],arg[9:])
                    self.captura_muestras(arg[1],arg[3],arg[6],arg[7],list(
    arg[9:])+[arg[2]])
310                 self.cierre(arg[8])
                return None
312
        def cierre_hilo_escritura(self):
314         Modulacion.queue.put(-1)
            Modulacion.event.wait()
316
            Modulacion.queue_open_loop.put(-1)
318         Modulacion.event_open_loop.wait()

320         #Esperamos a que el hilo acabe
            self.write_thread.join()
322         self.open_loop_thread.join()

324         return

326 class Puro(Modulacion):

328     Rl_2600=440
        sensorPin2600 = 'P9_38'
330     heatPin2600 = ''
        path = "CAPTURAS/PURO"
332     strings = ['Valor(mV):','Rs(ohmios):','Temperatura(%5V): 100 inst_captura
    :']

334     def __init__(self,config_experiment='',config_platform='',**kwargs):
            super().__init__(config_experiment,config_platform,Puro.strings
    ,**kwargs)
336         Puro.Rl_2600 = int(config_platform[tc.SRSTCE]) if config_platform
    [tc.SRSTCE] != '' else 440
            Puro.sensorPin2600 = config_platform[tc.SRDPORT] if
    config_platform[tc.SRDPORT] != '' else 'P9_38'
338         Puro.heatPin2600 = config_platform[tc.SHTPORT] if config_platform
    [tc.SHTPORT] != '' else ''
            Puro.path = config_platform[tc.SSDFOLDER] if config_platform[tc.
```

```
    SSDFOLDER] != '' else 'CAPTURAS/PURO'
340                 self.muestras = 0

342         def valor_sensor(self,count,string,gas):
                    """
344                 Captura muestras sin usar ninguna tecnica.
                    Realiza la toma de medidas sin usar ninguna tecnica de modulacion
346                 Parametros:
                    count -- contador que indica las muestras que hemos tomado
348                 string -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
                    gas -- array que contiene los gases de la muestra que se captura
350                 """

352                 #Lectura nariz
                    time_ini = time.time()
354                 value=0
                    ADC.read(Puro.sensorPin2600) #la primera medida es erronea por el
     bug
356
                    # Tomamos las diez muestras
358                 for i in range(Modulacion.NM):
                            value += ADC.read(Puro.sensorPin2600)*1800
360                         time.sleep(Modulacion.tsub)

362                 # Hacemos la media
                    valueTGS2600=value/Modulacion.NM
364                 instante_captura=datetime.now()

366                 #Se calcula el valor de la resistencia interna del sensor
                    RsTGS2600=((Modulacion.Vcc*Puro.Rl_2600)/(valueTGS2600/1000.))-
    Puro.Rl_2600
368
                    time_end = time.time()
370                 # Pasamos a la cola los valores que va a escribir
                    print("Envio esto:",[string,count,valueTGS2600,RsTGS2600,
    instante_captura,gas])
372                 Modulacion.queue.put([string,count,valueTGS2600,RsTGS2600,
    instante_captura,gas])

374                 return (time_end - time_ini)

376         def captura_odorante(self,vector_valvulas,vector_odorantes,n_muestras,
    string):
                    """
378                 Capturamos tantas muestras como se indique en los argumentos
                    Codigo comun para la captura de gases, que llama a la funcion de
    puro_TGS2600, abre y cierra las valvulas
380                 y mide el tiempo de captura de los gases.
                    Parametros:
382                 vector_odorantes -- el vector de los odorantes que van a captarse
     en esta apertura de valvulas
                    inicio -- el numero que marca el inicio de muestras que hay que
    coger
384                 fin -- el numero final de muestras que hay que coger
                    string -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
386                 """

388                 super().abrir_electrovalvulas(vector_valvulas)
                    for count in range(n_muestras):
390                         #time_ini = time.time()
                            value_sleep = self.valor_sensor(self.muestras,string,
```

```
    vector_odorantes)
392                         #time_end = time.time()
                            time.sleep(Modulacion.SLEEP - value_sleep)
394                         self.muestras+=1
                    super().cerrar_electrovalvulas()
396
                    return
398
        def file_TGS2600(self,ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio):
400                 self.f.write ('Algoritmo temperatura constante\n')
                    super().file_TGS2600(ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio)
402                 self.f.write('Valor de la resistencia de carga: '+str(Puro.
    Rl_2600)+'\n')

404         def apertura_escritura_ficheros(self,sw,samplesinicio,ct,nfile,nfolder,
    vsaodrs,arg_extra=None):
                    super().apertura_escritura_ficheros(Puro.path,sw,samplesinicio,ct
    ,nfile,nfolder,vsaodrs)
406                 self.g.writelines(str(sw)+' '+str(samplesinicio)+' '+str(ct)+' '+
    str(len(vsaodrs))+'\n')

408         def inicializar_hilos_puertos(self,suc,sw,samplesinicio,ct,nfile,nfolder,
    vsovs,vsaodrs,arg_extra=None):
                    super().inicializar_hilos_puertos(suc)
410                 ADC.setup()
                    self.file_TGS2600(tc.obtener_ruta(Puro.path,self.time_mark,
    nfolder,''),nfile+".txt",
412                             nfile+".dat","TyH"+nfile+".data",vsaodrs,suc,"5V",float(
    samplesinicio + (ct*(len(vsovs)+1)) + (len(vsovs)*sw)),sw,ct,samplesinicio)

414         def capturas_muestras_iniciales(self,samplesinicio,args_extra=None):
                    super().capturas_muestras_iniciales(samplesinicio)
416                 self.captura_odorante(Modulacion.capturas_iniciales,[4],
    samplesinicio,"Muestras_iniciales")

418         def captura_muestras(self,sw,ct,vsovs,vsaodrs,args_extra=None):
                    super().captura_muestras()
420
                    switch = sw if isinstance(sw,list) else [sw]*len(vsovs)
422                 for sw,vsov,vsaodr in zip(switch,vsovs,vsaodrs):
                            self.captura_odorante(Modulacion.capturas_iniciales,[4],
    ct,"Muestra_entre_gases")
424                         self.captura_odorante(vsov,vsaodr,sw,"Muestra_gases")

426                 self.captura_odorante(Modulacion.capturas_iniciales,[4],ct,"
    Muestra_entre_gases")
                    super().cerrar_electrovalvulas()
428
        def cierre(self,sle,arg_extra=None):
430                 super().cierre(Puro.heatPin2600)
                    self.muestras = 0
432                 time.sleep(sle)

434         def captura_datos(self):

436                 PWM.start(self.heatPin2600,100)
                    super().captura_datos()
438                 super().cierre_hilo_escritura()
                    return Puro.path
440
class Regresion(Modulacion):
```

```
442
        Rl_2600=440
444     sensorPin2600 = 'P9_40'
        heatPin2600 = 'P9_22'
446     path = "CAPTURAS/REGRESION"
        strings = ['Valor(mV):','Rs(ohmios):','Temperatura(%5V):','inst_captura:
    ','slope:','intercept:','r_value:','p_value:','std_err1:']
448
        def __init__(self,config_experiment='',config_platform='',**kwargs):
450             super().__init__(config_experiment,config_platform,Regresion.
    strings,**kwargs)
                self.tendencia = config_experiment[tc.STEND]
452             self.heat = config_experiment[tc.SHTSENSOR]
                self.x=[]
454             self.concentTGS2600=[]
                self.muestras = 0
456             Regresion.Rl_2600 = int(config_platform[tc.SRSTCE]) if
    config_platform[tc.SRSTCE] != '' else 440
                Regresion.sensorPin2600 = config_platform[tc.SRDPORT] if
    config_platform[tc.SRDPORT] != '' else 'P9_40'
458             Regresion.heatPin2600 = config_platform[tc.SHTPORT] if
    config_platform[tc.SHTPORT] != '' else 'P9_22'
                Regresion.path = config_platform[tc.SSDFOLDER] if config_platform
    [tc.SSDFOLDER] != '' else 'CAPTURAS/REGRESION'
460
        def valor_sensor(self,count,temperature_TGS2600,string,opcion,gas,
    samplesinicio,tendencia):
462             """
                Captura muestras usando la tecnica de la regresion lineal
464             Realiza la toma de medidas usando como tecnica de modulacion la
    regresion lineal
                de la temperatura en funcion de la muestra obtenida en el sensor.
466             Parametros:
                count -- contador que indica las muestras que hemos tomado
468             temperature_TGS2600 -- temperatura inicial a la que calentamos el
     sensor
                string -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
470             opcion -- 1 para las SAMPLESINICIO muestras, 2 para capturas
    tantas muestras como hayamos indicado
                gas -- array que contiene los gases de la muestra que se captura
472             """
                time_ini = time.time()
474             if opcion == 1:
                        PWM.set_duty_cycle(Regresion.heatPin2600,
    temperature_TGS2600)
476
                value=0
478             queda=0
                residuo=0
480             ADC.read(Regresion.sensorPin2600) #la primera medida es erronea
    por el bug


482
                for i in range(Modulacion.NM):
484                     value += ADC.read(Regresion.sensorPin2600)*1800
                        time.sleep(Modulacion.tsub)
486
                valueTGS2600=value/Modulacion.NM
488             instante_captura=datetime.now()

490             slope, intercept, r_value, p_value, std_err1 = 0,0,0,0,0
                #Adaptacion temperatura
```

```
492                 if opcion == 2:
                            # Solo usa una ventana de tantos datos como samples
    inicio tenga
494                         slope, intercept, r_value, p_value, std_err1 = stats.
    linregress(self.x[(count-samplesinicio):(count-1)],
                                    self.concentTGS2600[(count-samplesinicio):(count
    -1)])
496
                            temperature_TGS2600 = temperature_TGS2600 - (slope*
    tendencia)
498
                            if temperature_TGS2600 < 10.0:
500                                 temperature_TGS2600 = 10.0
                            elif temperature_TGS2600 > 90.0:
502                                 temperature_TGS2600 = 90.0

504                         print ("Los valores de la tendencia, el slope y la
    temperatura son: "+str(tendencia)+", "+str(slope)+" y "+str(
    temperature_TGS2600))
                            #Reset de setup PWM
506                         PWM.set_duty_cycle(Regresion.heatPin2600,
    temperature_TGS2600)

508                 #Se calcula el valor de la resistencia interna del sensor
                    RsTGS2600=((Modulacion.Vcc*Regresion.Rl_2600)/(valueTGS2600
    /1000.))-Regresion.Rl_2600
510
                    self.x.append(count)
512                 self.concentTGS2600.append(valueTGS2600)
                    time_end = time.time()
514
                    Modulacion.queue.put([string,count,valueTGS2600,RsTGS2600,
    temperature_TGS2600,instante_captura,slope, intercept, r_value, p_value,
    std_err1,gas])
516                 return (time_end-time_ini)

518         def captura_odorante(self,vector_valvulas,vector_odorantes,n_muestras,
    string,opcion,heat2600,samplesinicio,tendencia):

520                 """
                    Capturamos tantas muestras como se indique en los argumentos
522                 Codigo comun para la captura de gases, que llama a la funcion de
    puro_TGS2600, abre y cierra las valvulas
                    y mide el tiempo de captura de los gases.
524                 Parametros:
                    vector_odorantes -- el vector de los odorantes que van a captarse
     en esta apertura de valvulas
526                 inicio -- el numero que marca el inicio de muestras que hay que
    coger
                    fin -- el numero final de muestras que hay que coger
528                 string -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
                    opcion -- 1 para las SAMPLESINICIO muestras, 2 para capturas
    tantas muestras como hayamos indicado
530                 heat2600 -- temperatura inicial a la que calentamos el sensor
                    """
532                 super().abrir_electrovalvulas(vector_valvulas)
                    for count in range(n_muestras):
534                         #time_ini = time.time()
                            value_sleep = self.valor_sensor(self.muestras,heat2600,
    string,opcion,vector_odorantes,samplesinicio,tendencia)
536                         #time_end = time.time()
                            time.sleep(Modulacion.SLEEP - value_sleep)
```

```
538                         self.muestras+=1
                    super().cerrar_electrovalvulas()
540                 ############
                    Modulacion.queue_open_loop.put([self.concentTGS2600[-n_muestras
    :],vector_odorantes])
542                 ###########

544         def file_TGS2600(self,ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio,tendencia)
    :
                    self.f.write('Algoritmo temperatura variable y codificacion en
    amplitud\n')
546                 super().file_TGS2600(ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio)
                    self.f.write('Pin PWM de calentamiento sensor: ' + str(self.
    heatPin2600)+'\n')
548                 self.f.write('Valor de la tendencia: ' + str(tendencia)+'\n')
                    self.f.write('Pin ADC sensor: ' + str(self.sensorPin2600)+'\n')
550                 self.f.write('Valor de la resistencia de carga: '+str(Regresion.
    Rl_2600)+'\n')

552         def apertura_escritura_ficheros(self,sw,samplesinicio,ct,nfile,nfolder,
    vsaodrs,arg_extra=None):
                    super().apertura_escritura_ficheros(Regresion.path,sw,
    samplesinicio,ct,nfile,nfolder,vsaodrs)
554                 self.g.writelines(str(sw)+' '+str(samplesinicio)+' '+str(ct)+' '+
    str(len(vsaodrs))+'\n')
                    return
556

         def inicializar_hilos_puertos(self,suc,sw,samplesinicio,ct,nfile,nfolder,
    vsovs,vsaodrs,arg_extra=None):
558                 super().inicializar_hilos_puertos(suc)
                    ADC.setup()
560                 PWM.start(Regresion.heatPin2600,arg_extra[0])

562                 self.file_TGS2600(tc.obtener_ruta(Regresion.path,self.time_mark,
    nfolder,''),nfile+".txt",nfile+".dat","TyH"+nfile+".data",
                            vsaodrs,suc,str(arg_extra[0]*0.01*5)+"V",float(
    samplesinicio + (ct*(len(vsovs)+1)) + (len(vsovs)*sw)),sw,ct,samplesinicio,
    arg_extra[1])
564

         def capturas_muestras_iniciales(self,samplesinicio,args_extra=None):
566                 super().capturas_muestras_iniciales(samplesinicio)
                    self.captura_odorante(Modulacion.capturas_iniciales,[4],
    samplesinicio,"Muestras_iniciales",1,args_extra[0],samplesinicio,args_extra
    [1])
568

         def captura_muestras(self,sw,ct,vsovs,vsaodrs,args_extra=None):
570                 super().captura_muestras()

572                 switch = sw if isinstance(sw,list) else [sw]*len(vsovs)
                    for sw,vsov,vsaodr in zip(switch,vsovs,vsaodrs):
574                         self.captura_odorante(Modulacion.capturas_iniciales,[4],
    ct,"Muestra_entre_gases",2,args_extra[0],args_extra[2],args_extra[1])
                            self.captura_odorante(vsov,vsaodr,sw,"Muestra_gases",2,
    args_extra[0],args_extra[2],args_extra[1])
576

                    self.captura_odorante(Modulacion.capturas_iniciales,[4],ct,"
    Muestra_entre_gases",2,args_extra[0],args_extra[2],args_extra[1])
578                 super().cerrar_electrovalvulas()

580         def cierre(self,sle,arg_extra=None):
                    super().cierre(self.heatPin2600)
```

```
582                 self.x=[]
                    self.concentTGS2600=[]
584                 self.muestras = 0
                    time.sleep(sle)
586
        def captura_datos(self):
588
                    super().captura_datos(self.heat,self.tendencia)
590                 super().cierre_hilo_escritura()
                    return Regresion.path
592
class Martinelli(Modulacion):
594
        Rl_2600=27000
596     sensorPin555 = 'P9_12'
        heatPin2600 = 'P9_14'
598     path = "CAPTURAS/MARTINELLI"
        strings = ['Count: ',' t_up: ',' t_down: ',' Durancion pulso: ',' Heat
    (%3.3V): ',' Instante captura inicial: ',' Instante captura final: ', ' Gas(es
    ) captados e identidicadores:']
600

602     def __init__(self,config_experiment='',config_platform='',**kwargs):
                    super().__init__(config_experiment,config_platform,Martinelli.
    strings,**kwargs)
604                 self.execution_modes = config_experiment[tc.SMMEXE]
                    self.temperature_modes = config_experiment[tc.SMMTEM]
606                 self.Tmins = config_experiment[tc.SMTMIN]
                    self.Tmaxs = config_experiment[tc.SMTMAX]
608                 self.thread_temp = None
                    Martinelli.Rl_2600 = int(config_platform[tc.SRSTCE]) if
    config_platform[tc.SRSTCE] != '' else 27000
610                 Martinelli.sensorPin555 = config_platform[tc.SRDPORT] if
    config_platform[tc.SRDPORT] != '' else 'P9_12'
                    Martinelli.heatPin2600 = config_platform[tc.SHTPORT] if
    config_platform[tc.SHTPORT] != '' else 'P9_14'
612                 Martinelli.path = config_platform[tc.SSDFOLDER] if
    config_platform[tc.SSDFOLDER] != '' else 'CAPTURAS/MARTINELLI'
                    GPIO.setup(Martinelli.sensorPin555,GPIO.IN)
614
        def martinelli_thread_temperature(self,fd,Tmin,Tmax):
616
                    tf = currentThread()
618                 value,secs,times,temperature_TGS2600,flag = 1,-1,0,Tmin-1,False

620                 while getattr(tf,"global_variable",True):
                            while getattr(tf,"partial_variable",True):
622                                 time_ini = time.time()
                                    if times %8 == 0 and flag == False and times >
    0:
624                                         value*=-1
                                    flag = True
626                                 temperature_TGS2600+=value
                                    secs+=1
628                                 if temperature_TGS2600 > Tmax:
                                            temperature_TGS2600 = Tmax
630                                 if temperature_TGS2600 < Tmin:
                                            temperature_TGS2600 = Tmin
632                                 PWM.set_duty_cycle(Martinelli.heatPin2600,
    temperature_TGS2600)
                                    fd.writelines(str(times)+' '+str(secs)+' '+str(
    temperature_TGS2600)+' '+str(datetime.now())+'\n')
634                                 fd.flush()
```

```
                                time.sleep(Modulacion.SLEEP - (time.time() -
    time_ini))
636
                        if flag == True:
638                             times+=1
                                flag = False
640

642             fd.close()

644     def adjust_temperature_TGS(self,mode_temperature,periodo,subperiodo,Tmin,
    Tmax):

646             if mode_temperature == 1:
                        if periodo == 0:
648                             return Tmin
                        else:
650                             return Tmax
                else:
652                     if periodo == 0:
                                return Tmax - (((Tmax-Tmin)/8)*(subperiodo+1))
654                     else:
                                return Tmin + (((Tmax-Tmin)/8)*(subperiodo+1))
656
        def valor_sensor(self,string, gas, mode_temperature, Tmin, Tmax):
658             """
                Captura muestras usando la tecnica de Martinelli
660             Realiza la toma de medidas usando como tecnica la descrita por
    Martinelli
                Parametros:
662             stringA -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
                stringB -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
664             gas -- array que contiene los gases de la muestra que se captura
                """
666
                count = 0
668             sumatorio = 0

670             #Realizamos dos tomas de 8 cada una,16 en total, variando
    unicamente la temperatura
                for i in range(2):
672                     #Cogemos 8 muestras, las escribimos en un fichero, y
    luego otras 8
                        for j in range(8):
674
                                #Fijamos la temperatura
676                             temperature_TGS2600 = self.adjust_temperature_TGS
    (mode_temperature,i,j,Tmin,Tmax)
                                if mode_temperature != 3:
678                                     PWM.set_duty_cycle(self.heatPin2600,
    temperature_TGS2600)

680                             #Espera pulso de caida
                                instante_captura_ini=datetime.now() #Lo pongo
    afuera para que no interfiera en la captura.
682                             GPIO.wait_for_edge(self.sensorPin555, GPIO.
    FALLING,500000) # El numero es un timeout de 500000 milisegundos
                                ini_pulso = time.time()
684                             if mode_temperature == 3:
                                        self.thread_temp.partial_variable = True
686
```

```
                                GPIO.wait_for_edge(self.sensorPin555, GPIO.RISING
,500000)
688                             ini_up = time.time()

690                             GPIO.wait_for_edge(self.sensorPin555, GPIO.
FALLING,500000)       #Espera pulso de bajada
                                fin_pulso = time.time()
692                             if mode_temperature == 3:
                                        self.thread_temp.partial_variable = False
694                             instante_captura=datetime.now()

696                             time_down = ini_up - ini_pulso
                                time_up = fin_pulso - ini_up              #Duracion
 del pulso en estado alto
698                             time_pulso = fin_pulso - ini_pulso        #Duracion
 del pulso completo

700                             sumatorio+=time_pulso
                                Modulacion.queue.put([string,count,time_up,
time_down,temperature_TGS2600,instante_captura_ini,instante_captura,gas])
702
                                count+=1
704
            return sumatorio
706
    def captura_odorante(self,string,i,vector_odorantes,mode_execution,
mode_temperature,Tmin,Tmax):
708
            """
710         Capturamos los 16 pulsos del gas.
            Codigo comun para la captura de gases, que llama a la funcion de
martinelli, abre y cierra las valvulas
712         y mide el tiempo de captura de los gases.
            Parametros:
714         stringA -- cadena que indica si una muestra es inicial o no, se
usa al imprimir en los ficheros
            stringB -- cadena que indica si una muestra es inicial o no, se
usa al imprimir en los ficheros
716         stringC -- cadena que indica si una muestra es inicial o no, se
usa al imprimir en los ficheros
            i -- contador que muestra cuantos veces hemos capturado muestras
718         vector_odorantes -- el vector de los odorantes que van a captarse
 en esta apertura de valvulas
            """
720
            time_martinelli = self.valor_sensor(string,vector_odorantes,
mode_temperature,Tmin,Tmax)
722         instante_captura = datetime.now()
            print ('\n'+string+str(i)+' Duracion de los k=16 pulsos:'+str(
time_martinelli)+' >> '+str(instante_captura)+'\n')
724         wline_f = string+str(i)+' Duracion de los k=16 pulsos:'+str(
time_martinelli)+' >> '+str(instante_captura)+'\n'
            self.f.writelines(wline_f)
726         self.f.flush()

728         if mode_execution == 1:
                    return time_martinelli
730         else:
                    return 1
732
    def file_TGS2600(self,ruta,nameFile,nameFile_data1,nameFile_data2,
nameFile_tyh,vec_open_valve,succion,heat2600,tiempo,switch,tespera,
samplesinicio,mode_execution,mode_temperature,Tmin,Tmax):
```

```
734                 self.f.write('Algoritmo temperatura variable y codificacion en
    frecuencia\n')
                    super().file_TGS2600(ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio)
736                 self.f.write ('Nombre del fichero de datos 2: ' + str(
    nameFile_data2) + '\n')
                    self.f.write('Pin PWM de calentamiento sensor: ' + str(self.
    heatPin2600)+ '\n')
738                 self.f.write('Pin ADC sensor: ' + str(self.sensorPin555) + '\n')
                    self.f.write('Modo de ejecucion: ' + str(mode_execution)+'\n')
740                 self.f.write('Modo de cambio de temperatura: ' + str(
    mode_temperature)+'\n')
                    self.f.write('Temperatura minima del sensor: '+str(Tmin)+'\n')
742                 self.f.write('Temperatura maxima del sensor: '+str(Tmax) + '\n')

744         def apertura_escritura_ficheros(self,sw,samplesinicio,ct,nfile,nfolder,
    vsaodrs,arg_extra):

746                 super().apertura_escritura_ficheros(Martinelli.path,sw,
    samplesinicio,ct,nfile,nfolder,vsaodrs)
                    self.g.writelines(str(arg_extra[0]) + ' ' + str(arg_extra[1]) + '
     ' + str(sw)+' '+str(samplesinicio)+' '+str(ct)+' '+str(len(vsaodrs))+' '+str(
    arg_extra[2])+' '+str(arg_extra[3])+'\n')
748
         def inicializar_hilos_puertos(self,suc,sw,samplesinicio,ct,nfile,nfolder,
    vsovs,vsaodrs,arg_extra):
750                 #Llamada al thread de medida de temperatura y humedad
                    super().inicializar_hilos_puertos(suc)
752                 PWM.start(Martinelli.heatPin2600,arg_extra[2])#,20000,0)

754                 self.file_TGS2600(tc.obtener_ruta(Martinelli.path,self.time_mark,
    nfolder,''),nfile+".txt",nfile+".dat",
                                      str(nfile+'_dat2_' + time.strftime("%a%d%b%Y-%HH%MM%SS",
    time.localtime())+".dat"),"TyH"+nfile+".data",vsaodrs,
756                                   suc,str(arg_extra[2]*0.01*5)+"V",float(samplesinicio + (
    ct*(len(vsovs)+1)) + (len(vsovs)*sw)),sw,ct,samplesinicio,
                                      arg_extra[0],arg_extra[1],arg_extra[2],arg_extra[3])
758
                    if arg_extra[1] == 3:
760                     self.thread_temp = Thread(target=
    martinelli_thread_temperature,args=(open(obtener_ruta(Martinelli.path,self.
    time_mark,nfolder,
                                        'dat')+"temperature_martinelli"+time.strftime("%a
    %d%b%Y-%HH%MM%SS",time.localtime())+".dat","w"),arg_extra[2],arg_extra[3]))
762                     self.thread_temp.global_variable = True
                        self.thread_temp.partial_variable = False
764                     self.thread_temp.start()

766         def capturas_muestras_iniciales(self,samplesinicio,args_extra):
                    super().capturas_muestras_iniciales(samplesinicio)
768
                    #TOMAMOS X MUESTRAS DE INICIO
770                 super().abrir_electrovalvulas(Modulacion.capturas_iniciales)
                    for i in range(samplesinicio):
772                     self.captura_odorante('Muestra_ini',i,Modulacion.
    capturas_iniciales,args_extra[0],args_extra[1],args_extra[2],args_extra[3])
                    super().cerrar_electrovalvulas()
774
         def sub_captura_muestras(self,w,vsov,vsaodr,sw,mode_execution,
    mode_temperature,Tmin,Tmax):
776
                    tiempo_valve = 0
778                 super().abrir_electrovalvulas(vsov)
```

```
                while tiempo_valve < sw: # Capturas tanto tiempo como conmutacion
    desees
780                     tiempo_valve +=self.captura_odorante('Muestra_odorante',w
    ,vsaodr,mode_execution,mode_temperature,Tmin,Tmax)
                        print ('Tiempo acumulado = '+str(tiempo_valve))
782             super().cerrar_electrovalvulas()
                w+=1
784
                return w
786
        def captura_muestras(self,sw,ct,vsovs,vsaodrs,args_extra):
788             super().captura_muestras()

790             w = 0
                for vsov,vsaodr in zip(vsovs,vsaodrs):
792
                    w = self.sub_captura_muestras(w,Modulacion.
    capturas_iniciales,Modulacion.capturas_iniciales,
794                                 ct,args_extra[0],args_extra[1],args_extra[2],
    args_extra[3])

796                 w = self.sub_captura_muestras(w,vsov,vsaodr,sw,args_extra
    [0],args_extra[1],args_extra[2],args_extra[3])

798                 w = self.sub_captura_muestras(w,Modulacion.
    capturas_iniciales,Modulacion.capturas_iniciales,
                                ct,args_extra[0],args_extra[1],args_extra[2],
    args_extra[3])
800
                return
802
        def cierre(self,sle,arg_extra):
804             super().cierre(self.heatPin2600)
                self.k.close()
806             if arg_extra[1] == 3:
                        self.thread_temp.global_variable = False
808                     self.thread_temp.join()
                time.sleep(sle)
810
        def captura_datos(self):
812
                super().captura_datos(self.execution_modes,self.temperature_modes
    ,self.Tmins,self.Tmaxs)
814             super().cierre_hilo_escritura()
                return Martinelli.path
816

818 class MPID(Modulacion): #ModulationPID

820         Rl_2600=440
            sensorPin2600 = 'P9_40'
822         heatPin2600 = 'P9_22'
            path = "CAPTURAS/MPID"
824         strings = ['Target(mV):','Valor(mV):','Rs(ohmios):','Temperatura(%5V):','
    Temperatura_PID(%5V):','inst_captura: ']

826         def __init__(self,config_experiment='',config_elems='',**kwargs):
                super().__init__(config_experiment,config_elems,MPID.strings,**
    kwargs)
828             self.periods = config_experiment[tc.SPIDPERIOD]
                self.heat = config_experiment[tc.SHTSENSOR]
830             self.Kp = list(np.array(config_experiment[tc.SALPHA])*0.6)
                self.Kd = list(np.array(self.periods)*0.125)
```

```
832                 self.Ki = list(np.array(self.periods)*0.5)
                    self.temperature_Max_Upper_Bound = config_experiment[tc.
    SMAXLIMITUPPERBOUND]
834                 self.temperature_Min_Upper_Bound = config_experiment[tc.
    SMINLIMITUPPERBOUND]
                    self.temperature_Max_Lower_Bound = config_experiment[tc.
    SMAXLIMITLOWERBOUND]
836                 self.temperature_Min_Lower_Bound = config_experiment[tc.
    SMINLIMITLOWERBOUND]
                    self.max_peak_value = config_experiment[tc.SMAXPKVAL]
838                 self.min_peak_value = config_experiment[tc.SMINPKVAL]
                    self.target = None # Target que hay que seguir
840                 self.max_value,self.min_value = 0,0 # Valores máximos y mínimos
    capturados por el sensor
                    self.max_PID_temp,self.min_PID_temp = 0,0 # Valores máximos y
    mínimos de la temperatura del sensor
842                 self.errorControl = 0.05
                    self.muestras = 0
844
                    MPID.Rl_2600 = int(config_elems[tc.SRSTCE]) if config_elems[tc.
    SRSTCE] != '' else 440
846                 MPID.sensorPin2600 = config_elems[tc.SRDPORT] if config_elems[tc.
    SRDPORT] != '' else 'P9_40'
                    MPID.heatPin2600 = config_elems[tc.SHTPORT] if config_elems[tc.
    SHTPORT] != '' else 'P9_22'
848                 MPID.path = config_elems[tc.SSDFOLDER] if config_elems[tc.
    SSDFOLDER] != '' else 'CAPTURAS/MPID'

850                 self.lastError,self.addError,self.temp = 0,0,0

852         def crear_target(self,Vmax, Vmin, periodo):

854                 A = (Vmax - Vmin)/2.0
                    B = (2*math.pi)/periodo
856
                    signal = []
858
                    for x in range(periodo+1):
860                         signal.append(round(((A*math.sin(B*x))+(A+Vmin)),3))

862                 self.target = signal[0:periodo]

864                 return

866         def recalcular_target(self,temperature_Max_Upper_Bound,
    temperature_Min_Upper_Bound,temperature_Max_Lower_Bound,
    temperature_Min_Lower_Bound,period):

868                 max_bound,min_bound = 0,0
                    if (self.max_value - self.min_value) < 150.0:
870                         if (self.max_value > self.min_value):
                                if self.max_PID_temp >
    temperature_Max_Upper_Bound:
872                                     max_bound = math.ceil(self.max_value*0.9)
                                elif self.max_PID_temp <
    temperature_Min_Upper_Bound:
874                                     max_bound = math.ceil(self.max_value*1.1)
                                else:
876                                     max_bound = math.ceil(self.max_value)

878                             if self.min_PID_temp >
    temperature_Max_Lower_Bound:
                                    min_bound = math.ceil(max_bound*0.1)
```

```
880                                     elif self.min_PID_temp <
    temperature_Min_Lower_Bound:
                                             min_bound = math.ceil(self.min_value +
    max_bound*0.05)
882                                     else:
                                             min_bound = math.ceil(max_bound*0.5)
884                             else:
                                     max_bound,min_bound = math.ceil(self.min_value),
    math.ceil(self.min_value*0.3)
886
                                if min_bound < 100.0:
888                                     min_bound = 100.0
                                        if max_bound <= (min_bound*1.25):
890                                             max_bound = max_bound*1.25
                        else:
892                                if self.min_PID_temp > temperature_Max_Lower_Bound:
                                        min_bound = math.ceil(min(self.target)*0.95)
894                                elif self.min_PID_temp < temperature_Min_Lower_Bound:
                                        min_bound = math.ceil(min(self.target)*1.05)
896                                else:
                                        min_bound = self.min_value
898
                                if (max(self.target) - self.max_value) > 0:
900                                     max_bound = math.ceil(self.max_value)
                                elif self.max_PID_temp > temperature_Max_Upper_Bound:
902                                     max_bound = math.ceil(max(self.target)*0.98)
                                elif self.max_PID_temp < temperature_Min_Upper_Bound:
904                                     max_bound = math.ceil(max(self.target)*1.02)

906                     if max_bound < min_bound:
                                max_bound,min_bound = min_bound,max_bound
908
                        if max_bound > 1800.0:
910                             max_bound = 1800.0
                        if min_bound < 0.0:
912                             min_bound = 1800.0*0.01

914                     self.crear_target(max_bound,min_bound,period)

916                     return

918             def PID_controller(self, Kp, Kd, Ki, target_point, sensor_point,
    lastError, addError):

920                     error = round((target_point - sensor_point),3)
                        deltaTime = 1.0
922                     deltaError = error - lastError

924                     ep = error
                        ed = deltaError/deltaTime
926                     ei = addError + (error*deltaTime)

928                     output = Kp *(ep+(Kd*ed) + ((1/Ki)*ei))

930                     return output,error,ei

932             def valor_sensor(self,count,string, gas, subtarget, Kp, Kd, Ki,
    temperaturaMaxUpperBound, temperaturaMinLowerBound):
                        """
934                     Captura muestras usando la tecnica de Martinelli
                        Realiza la toma de medidas usando como tecnica la descrita por
    Martinelli
936                     Parametros:
```

```
            stringA -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
            stringB -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
            gas -- array que contiene los gases de la muestra que se captura
            """

            time_ini = time.time()
            value=0
            ADC.read(Puro.sensorPin2600) #la primera medida es erronea por el
     bug

            # Tomamos las diez muestras
            for i in range(Modulacion.NM):
                value += ADC.read(MPID.sensorPin2600)*1800
                time.sleep(Modulacion.tsub)

            # Hacemos la media
            valueTGS2600=value/Modulacion.NM
            instante_captura=datetime.now()

            output,error,ei = self.PID_controller(Kp,Kd,Ki,subtarget,
    valueTGS2600,self.lastError,self.addError)

            t1 = (output - max(self.target))*(((temperaturaMinLowerBound -
    temperaturaMaxUpperBound)/-max(self.target)))+temperaturaMaxUpperBound if
    error < -0.5 or error > 0.5 else 0

            if (self.temp < temperaturaMaxUpperBound or self.temp ==
    temperaturaMaxUpperBound) and (error < 0):
                temperaturaPID = self.temp - t1 if t1 > 0 else self.temp + t1
            elif ((self.temp > temperaturaMaxUpperBound) and (error < 0)) or
    ((self.temp < temperaturaMaxUpperBound) and (error > 0)):
                temperaturaPID = self.temp + t1
            else:
                temperaturaPID = self.temp
            if temperaturaPID > temperaturaMaxUpperBound:
                    temperaturaPID = temperaturaMaxUpperBound
            elif temperaturaPID < temperaturaMinLowerBound:
                    temperaturaPID = temperaturaMinLowerBound

            PWM.set_duty_cycle(MPID.heatPin2600, temperaturaPID)
            RSTGS = ((Modulacion.Vcc*MPID.Rl_2600)/(valueTGS2600/1000.0)) -
    MPID.Rl_2600

            time_end = time.time()
            Modulacion.queue.put([string,count,subtarget,valueTGS2600,RSTGS,
    self.temp,temperaturaPID,instante_captura,gas])

            self.lastError, self.addError, self.temp = error,ei,
    temperaturaPID
            return [time_end - time_ini,valueTGS2600]

    def calculate_min_max_value(self, subtarget, valueTGS2600, func, variable
    , tempPID, selftempPID):

            if subtarget == func(self.target):
                    e = subtarget*self.errorControl
                    selftempPID = tempPID
                    if not(valueTGS2600 >= (subtarget-e) and valueTGS2600 <=
    (subtarget+e)):
                            variable = valueTGS2600
                    else:
```

```
                                variable = func(self.target)
988
                return selftempPID, variable
990
        def captura_odorante(self,vector_valvulas,vector_odorantes,n_muestras,
    string,periodo, Kp, Kd, Ki,
992             temperature_Max_Upper_Bound,temperature_Min_Upper_Bound,
    temperature_Max_Lower_Bound,temperature_Min_Lower_Bound):
                """
994             Capturamos tantas muestras como se indique en los argumentos
                Codigo comun para la captura de gases, que llama a la funcion de
    puro_TGS2600, abre y cierra las valvulas
996             y mide el tiempo de captura de los gases.
                Parametros:
998             vector_odorantes -- el vector de los odorantes que van a captarse
     en esta apertura de valvulas
                inicio -- el numero que marca el inicio de muestras que hay que
    coger
1000            fin -- el numero final de muestras que hay que coger
                string -- cadena que indica si una muestra es inicial o no, se
    usa al imprimir en los ficheros
1002            """

1004            super().abrir_electrovalvulas(vector_valvulas)
                for count in range(n_muestras):
1006
                        if self.muestras %periodo == 0 and self.muestras != 0:
1008                        self.recalcular_target(temperature_Max_Upper_Bound,
    temperature_Min_Upper_Bound,temperature_Max_Lower_Bound,
    temperature_Min_Lower_Bound,periodo)

1010                    subtarget = self.target[self.muestras%periodo]
                        #time_ini = time.time()
1012                    ret_values = self.valor_sensor(self.muestras,string,
    vector_odorantes, subtarget, Kp, Kd, Ki, temperature_Max_Upper_Bound,
    temperature_Min_Lower_Bound)
                        #time_end = time.time()
1014                    time.sleep(Modulacion.SLEEP - ret_values[0])

1016                    self.max_PID_temp,self.max_value = self.
    calculate_min_max_value(subtarget, ret_values[1], max, self.max_value, self.
    temp, self.max_PID_temp)
                        self.min_PID_temp,self.min_value = self.
    calculate_min_max_value(subtarget, ret_values[1], min, self.min_value, self.
    temp, self.max_PID_temp)
1018                    self.muestras+=1

1020            super().cerrar_electrovalvulas()

1022            return

1024        def file_TGS2600(self,ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,
                heat2600,tiempo,switch,tespera,samplesinicio,period,Kp,Ki,Kd,
    temperature_Max_Upper_Bound,temperature_Min_Upper_Bound,
1026            temperature_Max_Lower_Bound,temperature_Min_Lower_Bound,
    maximum_peak_value,minimum_peak_value):

1028            self.f.write('Algoritmo modulacion por PID en lazo cerrado\n')
                super().file_TGS2600(ruta,nameFile,nameFile_data1,nameFile_tyh,
    vec_open_valve,succion,heat2600,tiempo,switch,tespera,samplesinicio)
1030            self.f.write('Valor de la resistencia de carga: '+str(MPID.
    Rl_2600)+'\n')
```

```
                self.f.write('Control Proporcional: '+str(Kp)+'\n')
1032            self.f.write('Control Integral: '+str(Ki)+'\n')
                self.f.write('Control Derivativo: '+str(Kd)+'\n')
1034            self.f.write('Pin PWM de calentamiento sensor: ' + str(self.
    heatPin2600)+ '\n')
                self.f.write('Pin ADC sensor: ' + str(self.sensorPin2600) + '\n')
1036            self.f.write('Periodo de la senial: '+str(period)+'\n')
                self.f.write('Limite maximo superior: '+str(
    temperature_Max_Upper_Bound)+'\n')
1038            self.f.write('Limite minimo superior: '+str(
    temperature_Min_Upper_Bound)+'\n')
                self.f.write('Limite maximo inferior: '+str(
    temperature_Max_Lower_Bound)+'\n')
1040            self.f.write('Limite minimo inferior: '+str(
    temperature_Min_Lower_Bound)+'\n')
                self.f.write('Valor pico inicial maximo: '+str(maximum_peak_value
    )+ '\n')
1042            self.f.write('Valor pico inicial minimo: '+str(minimum_peak_value
    )+ '\n')

1044        def apertura_escritura_ficheros(self,sw,samplesinicio,ct,nfile,nfolder,
    vsaodrs,arg_extra=None):
                super().apertura_escritura_ficheros(MPID.path,sw,samplesinicio,ct
    ,nfile,nfolder,vsaodrs)
1046            self.g.writelines(str(sw)+' '+str(samplesinicio)+' '+str(ct)+' '+
    str(len(vsaodrs))+'\n')

1048        def inicializar_hilos_puertos(self,suc,sw,samplesinicio,ct,nfile,nfolder,
    vsovs,vsaodrs,arg_extra=None):
                super().inicializar_hilos_puertos(suc)
1050            ADC.setup()
                PWM.start(MPID.heatPin2600,arg_extra[0])
1052            self.file_TGS2600(tc.obtener_ruta(MPID.path,self.time_mark,
    nfolder,''),nfile+".txt",nfile+".dat","TyH"+nfile+".data",
                        vsaodrs,suc,str(arg_extra[1]*0.01*5)+"V",float(
    samplesinicio + (ct*(len(vsovs)+1)) + (len(vsovs)*sw)),sw,ct,samplesinicio,
1054                    arg_extra[0],arg_extra[2],arg_extra[3],arg_extra[4],
    arg_extra[5],arg_extra[6],arg_extra[7],arg_extra[8],arg_extra[9], arg_extra
    [10])
                self.crear_target(arg_extra[9], arg_extra[10], arg_extra[0])
1056
            ############ FUNCION AUXILIAR ################
1058        # Calentamiento lineal
            def capturas_muestras_iniciales(self,samplesinicio,args_extra=None):
1060            super().capturas_muestras_iniciales(samplesinicio)

1062            super().abrir_electrovalvulas(Modulacion.capturas_iniciales)
                heatTGS,value = np.linspace(0,100,samplesinicio),0
1064            for count,heat in enumerate(heatTGS):
                    time_ini = time.time()
1066                PWM.set_duty_cycle(MPID.heatPin2600, heat)

1068                for i in range(Modulacion.NM):
                        value += ADC.read(MPID.sensorPin2600)*1800
1070                    time.sleep(Modulacion.tsub)

1072                # Hacemos la media
                    valueTGS2600=value/Modulacion.NM
1074                instante_captura=datetime.now()
                    RSTGS = ((Modulacion.Vcc*MPID.Rl_2600)/(valueTGS2600/1000.0)) -
    MPID.Rl_2600
1076                Modulacion.queue.put(["Muestras_iniciales",count,heat,
    valueTGS2600,RSTGS,heat,heat,instante_captura,[4]])
```

```
                print("Da time: ",Modulacion.SLEEP - (time.time() - time_ini))
1078            time.sleep(Modulacion.SLEEP - (time.time() - time_ini))
            super().cerrar_electrovalvulas()
1080        return False

1082    def captura_muestras(self,sw,ct,vsovs,vsaodrs,args_extra=None):
            super().captura_muestras()
1084
            switch = sw if isinstance(sw,list) else [sw]*len(vsovs)
1086        for sw,vsov,vsaodr in zip(switch,vsovs,vsaodrs):
                self.captura_odorante(Modulacion.capturas_iniciales,[4],
    ct,"Muestra_entre_gases",args_extra[0],args_extra[2],args_extra[3],args_extra
    [4],args_extra[5],args_extra[6],args_extra[7],args_extra[8])
1088            self.captura_odorante(vsov,vsaodr,sw,"Muestra_gases",
    args_extra[0],args_extra[2],args_extra[3],args_extra[4],args_extra[5],
    args_extra[6],args_extra[7],args_extra[8])

1090        self.captura_odorante(Modulacion.capturas_iniciales,[4],ct,"
    Muestra_entre_gases",args_extra[0],args_extra[2],args_extra[3],args_extra[4],
    args_extra[5],args_extra[6],args_extra[7],args_extra[8])
            super().cerrar_electrovalvulas()
1092
    def cierre(self,sle,arg_extra=None):
1094        super().cierre(MPID.heatPin2600)
            self.lastError,self.addError,self.temp = 0,0,0
1096        self.max_value,self.min_value = 0,0
            self.max_PID_temp,self.min_PID_temp = 0,0
1098        self.muestras = 0
            time.sleep(sle)
1100
    def captura_datos(self):
1102
            super().captura_datos(self.periods,self.heat,self.Kp,self.Kd,self
    .Ki,self.temperature_Max_Upper_Bound,
1104             self.temperature_Min_Upper_Bound,self.
    temperature_Max_Lower_Bound,self.temperature_Min_Lower_Bound,
                 self.max_peak_value,self.min_peak_value)
1106        super().cierre_hilo_escritura()
            return MPID.path
```

Listing C.1: Código principal del módulo de captura con los algoritmos de experimentación implementados.

D

Figuras

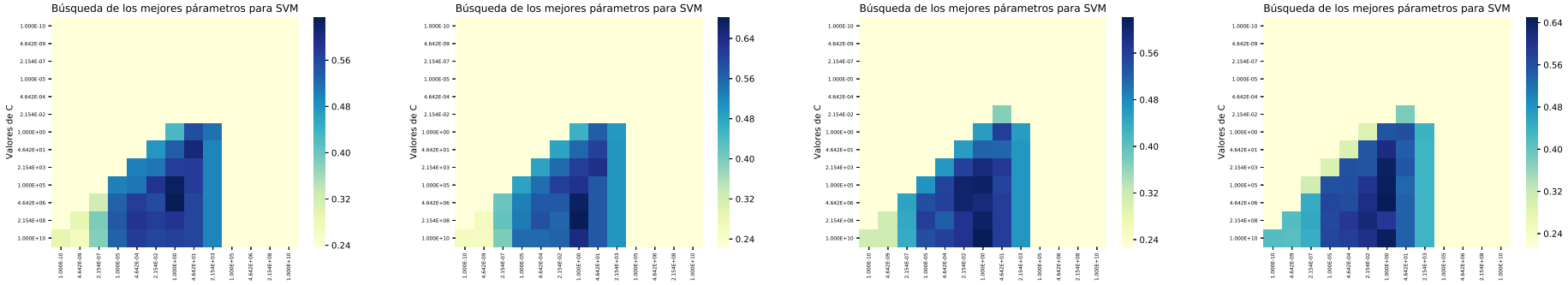


Figura D.1: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos del bulbo olfativo y las técnicas RQA. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

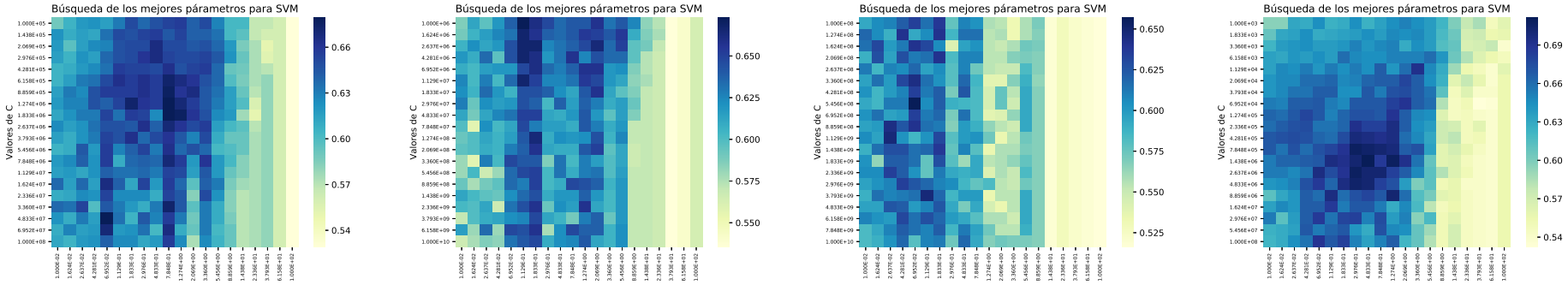


Figura D.2: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos del bulbo olfativo y las técnicas RQA. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

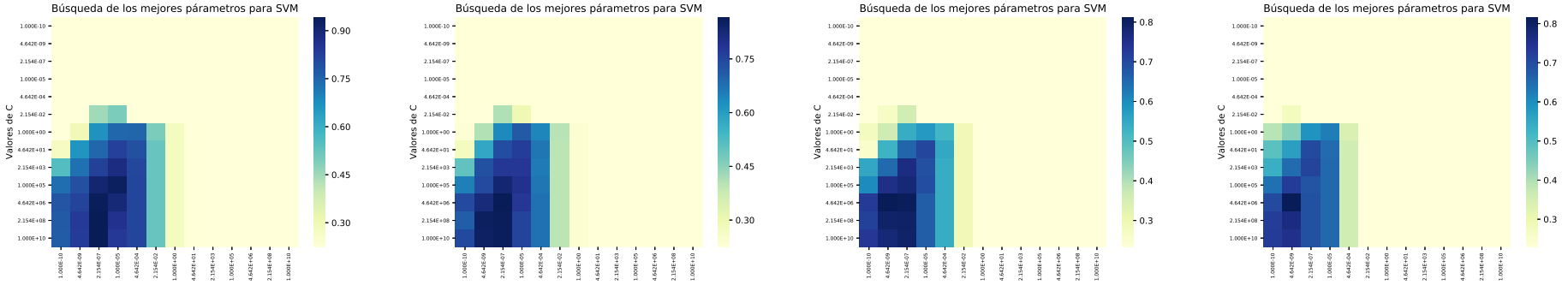


Figura D.3: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

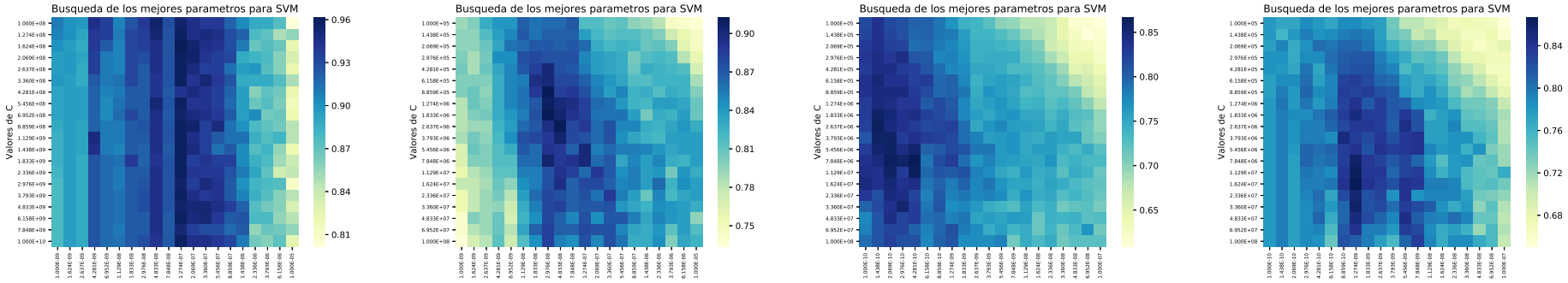


Figura D.4: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

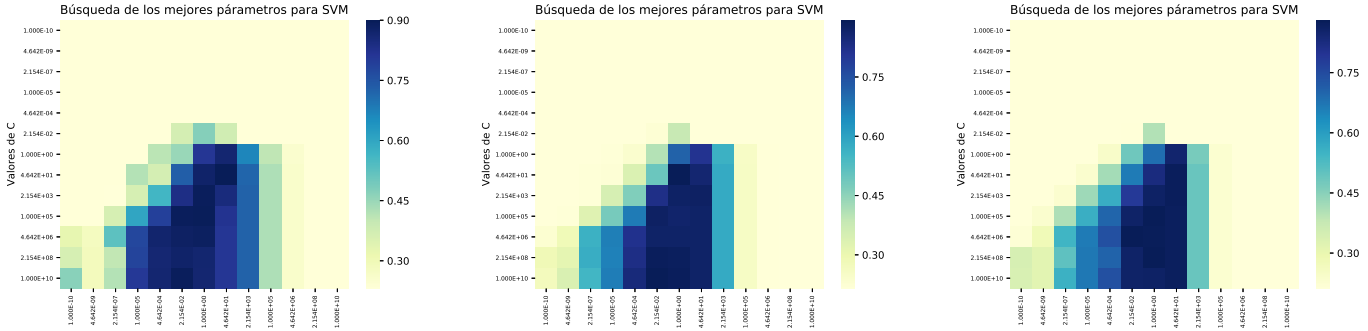


Figura D.5: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

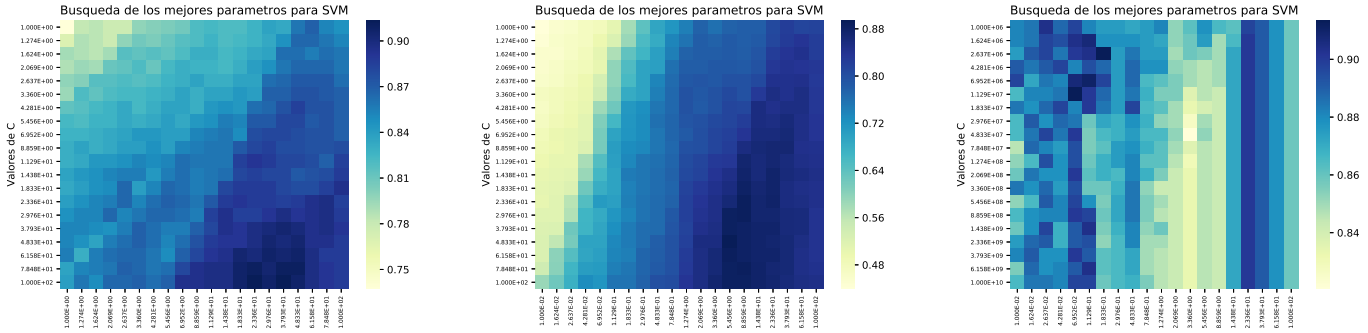


Figura D.6: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

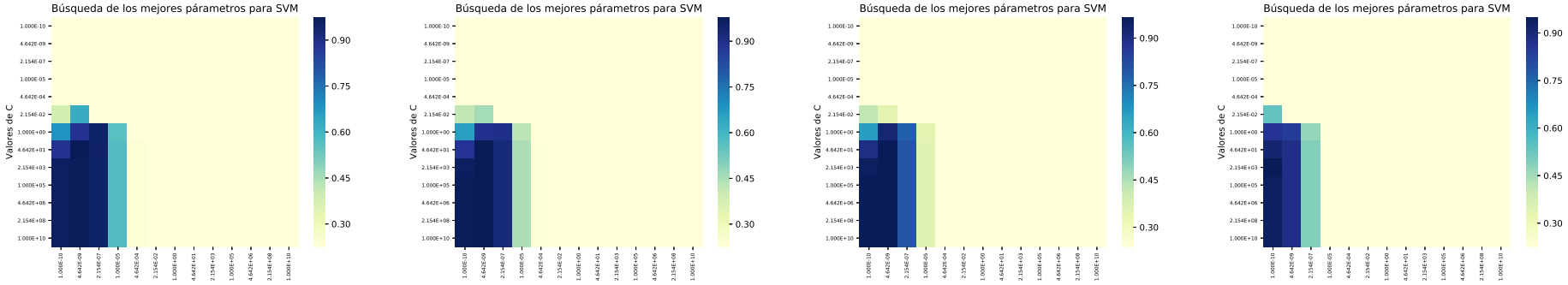


Figura D.7: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

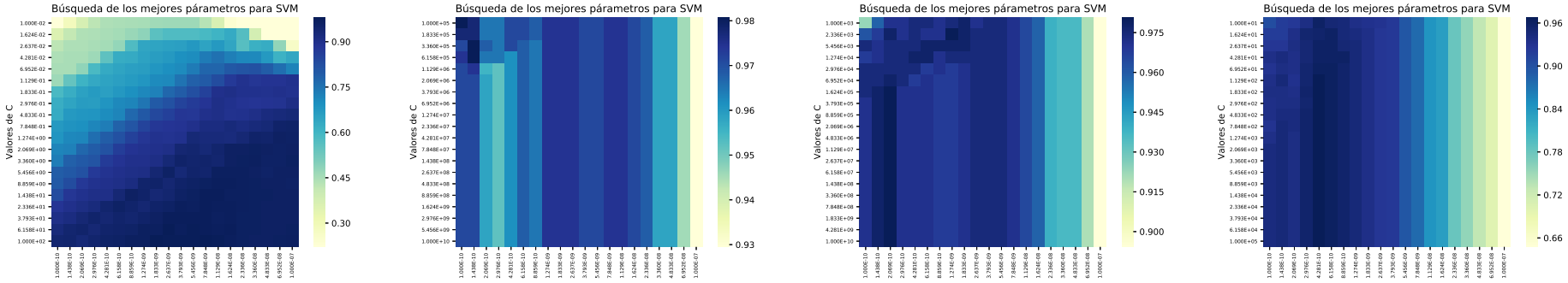


Figura D.8: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

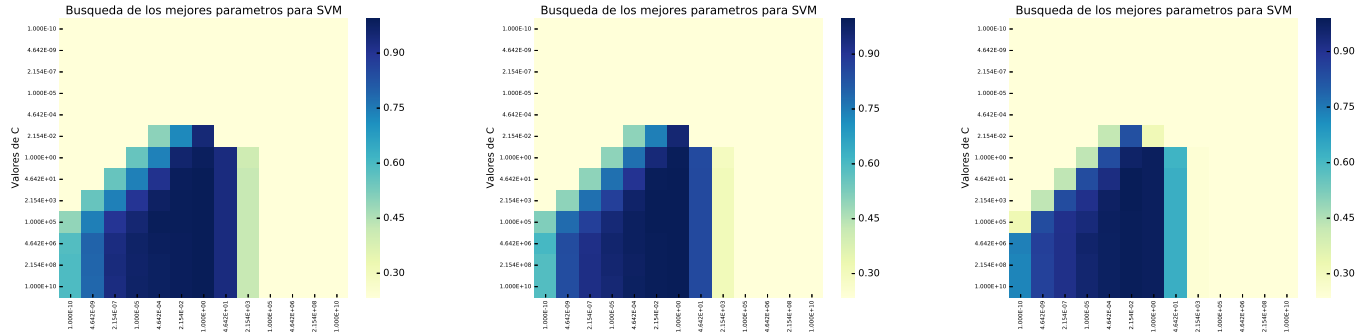


Figura D.9: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

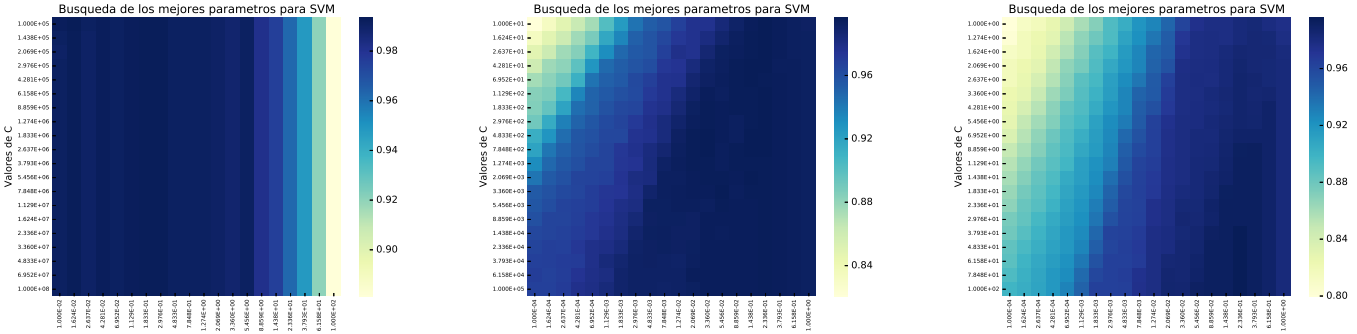


Figura D.10: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

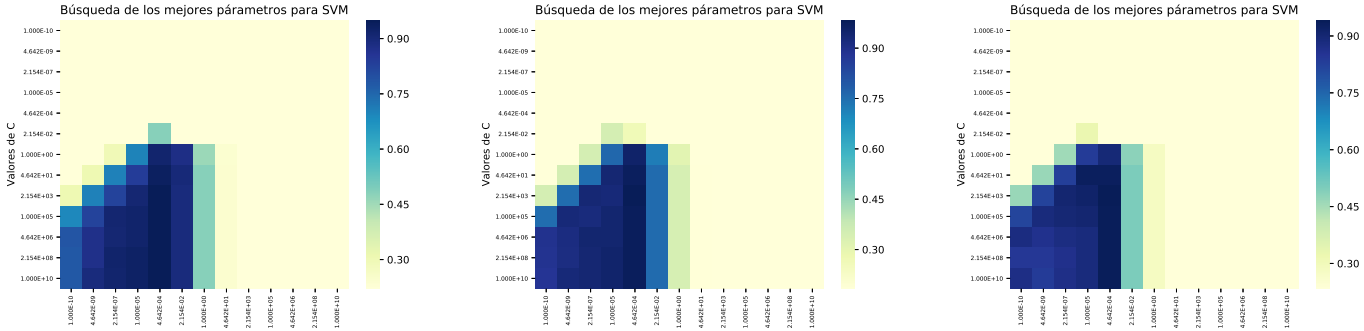


Figura D.11: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la temperatura. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

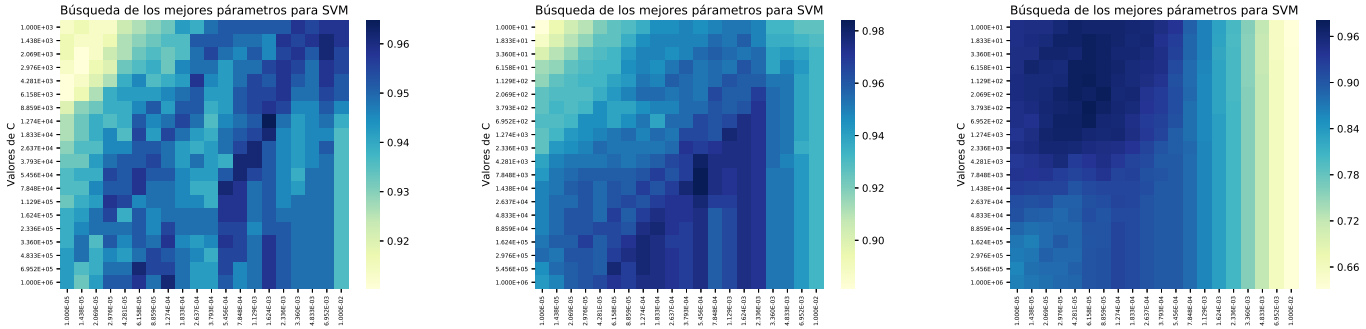


Figura D.12: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la temperatura. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

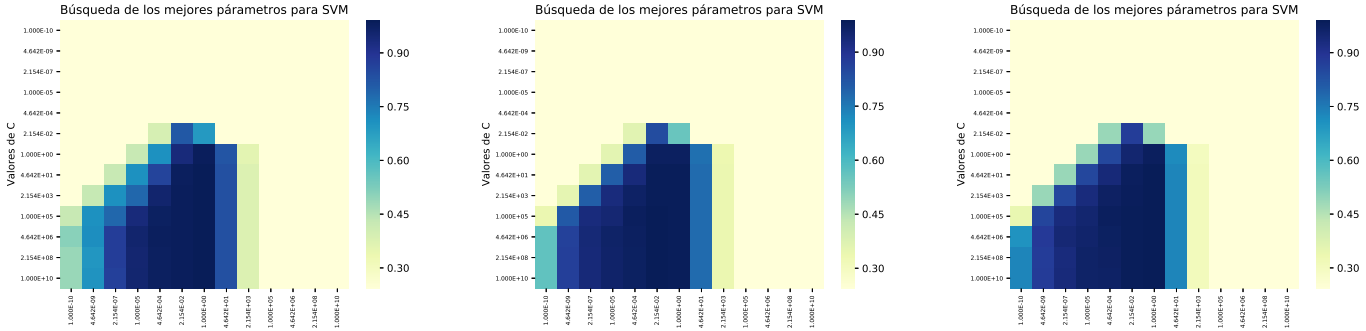


Figura D.13: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la temperatura habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

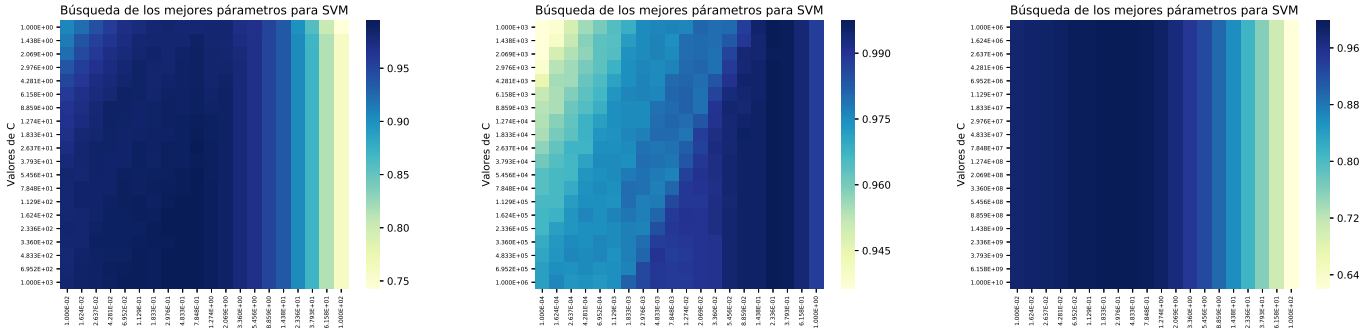


Figura D.14: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la temperatura habiendo aplicado DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

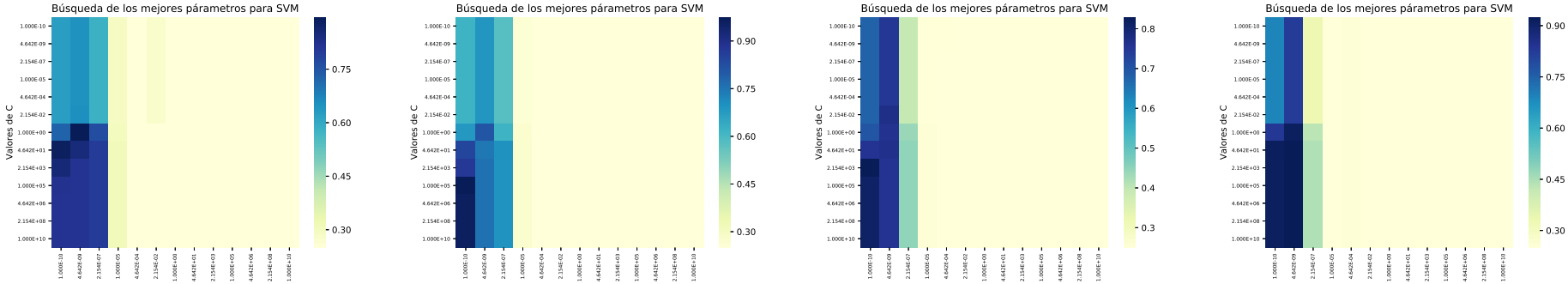


Figura D.15: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia habiendo aplicado el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

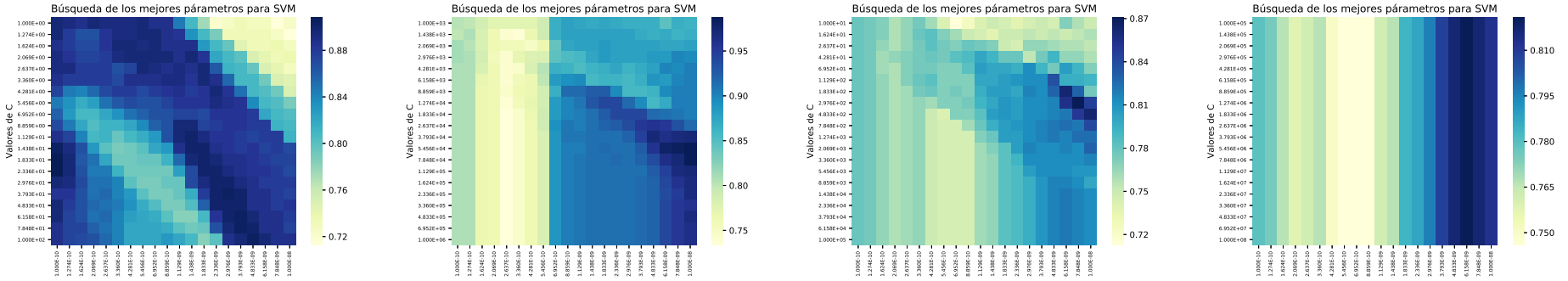


Figura D.16: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para la huella de la resistencia habiendo aplicado el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

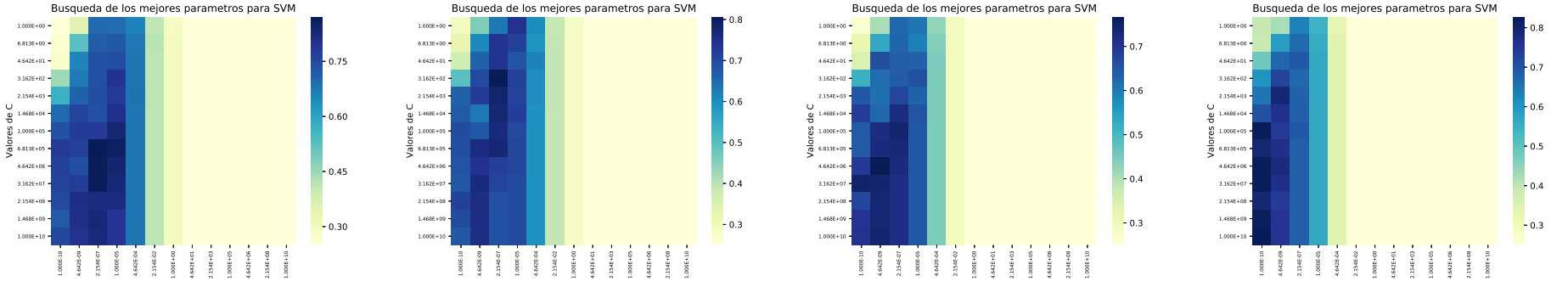


Figura D.17: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

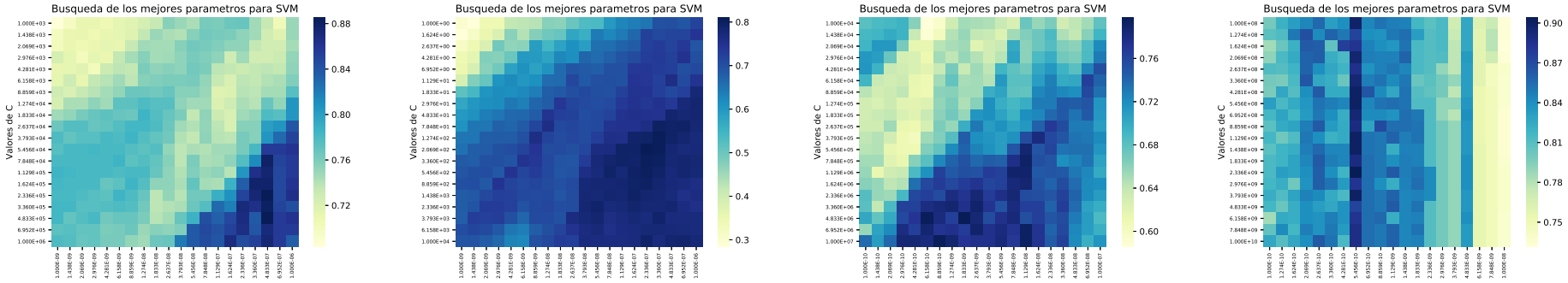


Figura D.18: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

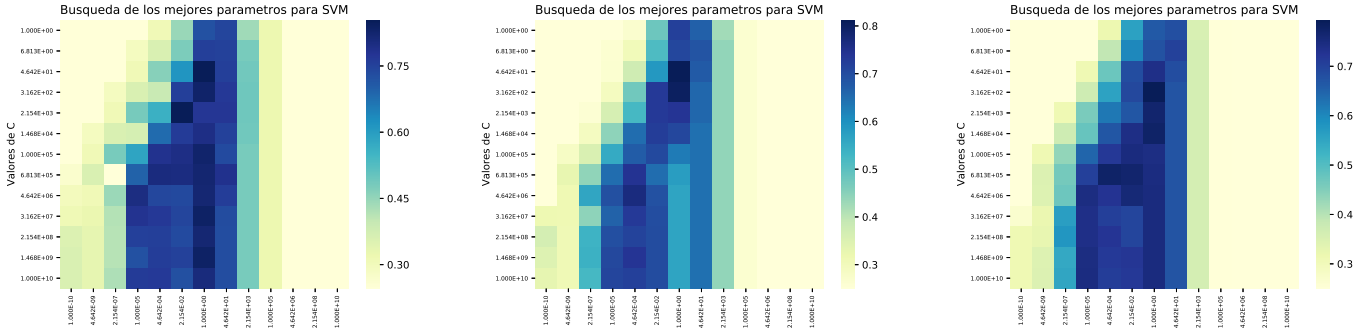


Figura D.19: Búsqueda en grid de grano grueso para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado el paradigma temporal y DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

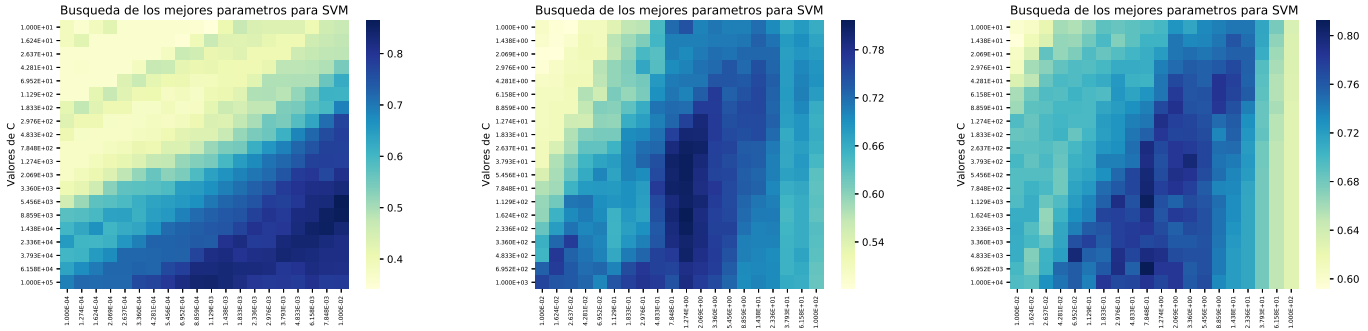


Figura D.20: Búsqueda en grid de grano fino para los valores C y γ con los que se obtendría un mejor resultado al clasificar, para los vectores de características obtenidos de EMA habiendo aplicado el paradigma temporal y DS. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

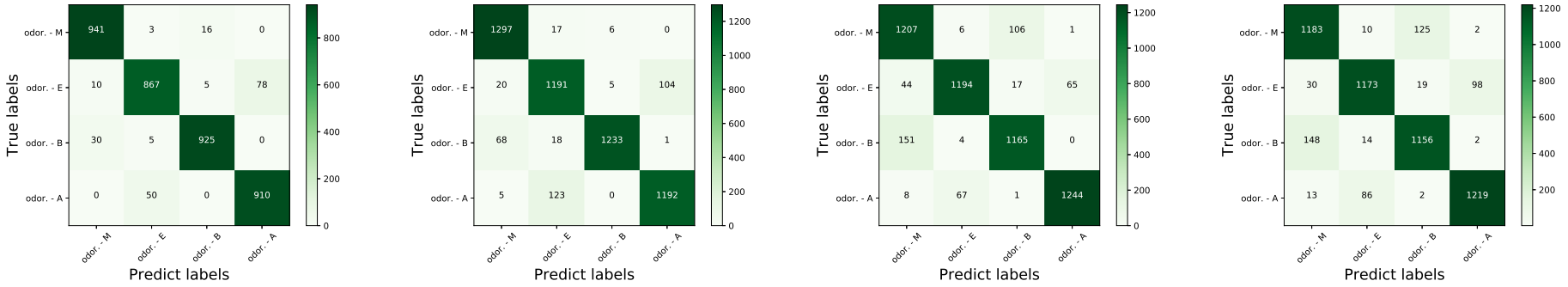


Figura D.21: Matriz de confusión obtenida de la clasificación mediante el uso de EMA para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

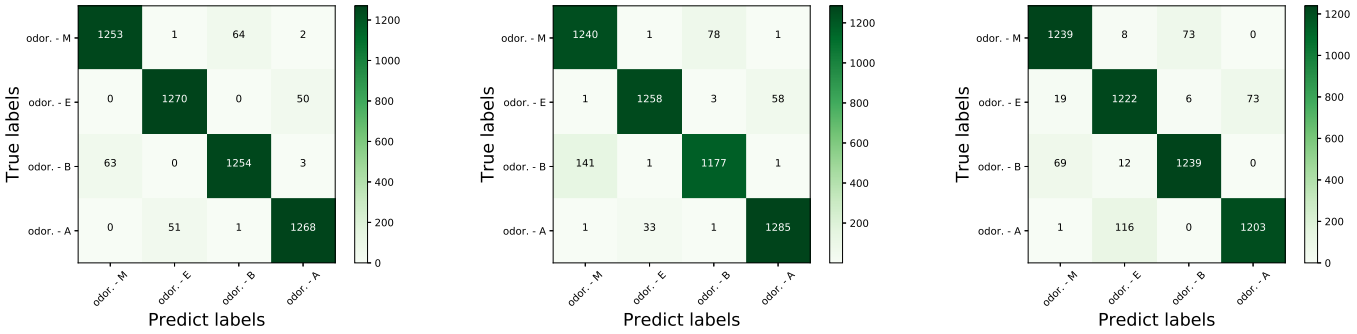


Figura D.22: Matriz de confusión obtenida de la clasificación mediante el uso de EMA y DS para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

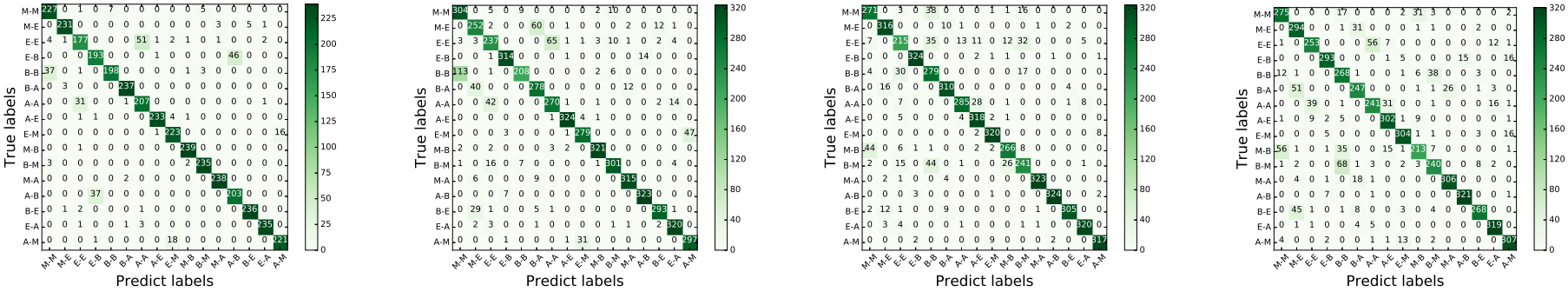


Figura D.23: Matriz de confusión obtenida de la clasificación mediante el uso de EMA para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

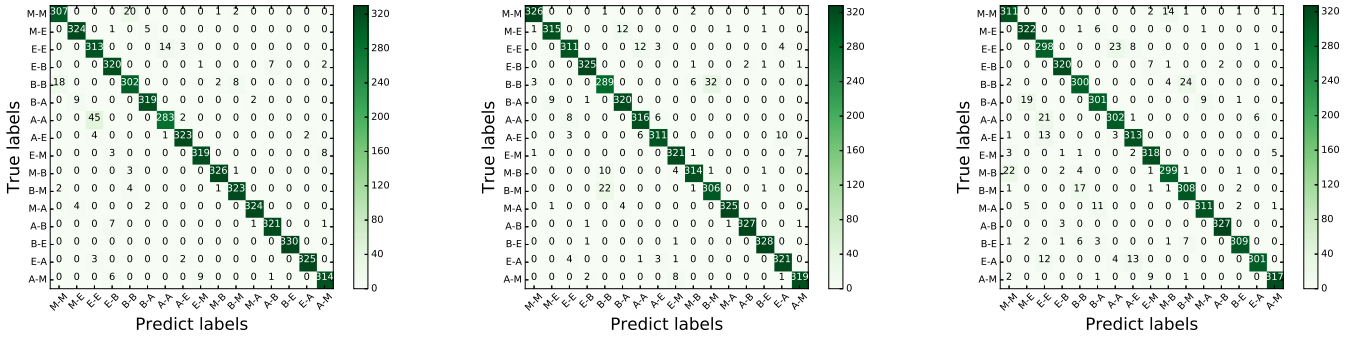


Figura D.24: Matriz de confusión obtenida de la clasificación mediante el uso de EMA y DS para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

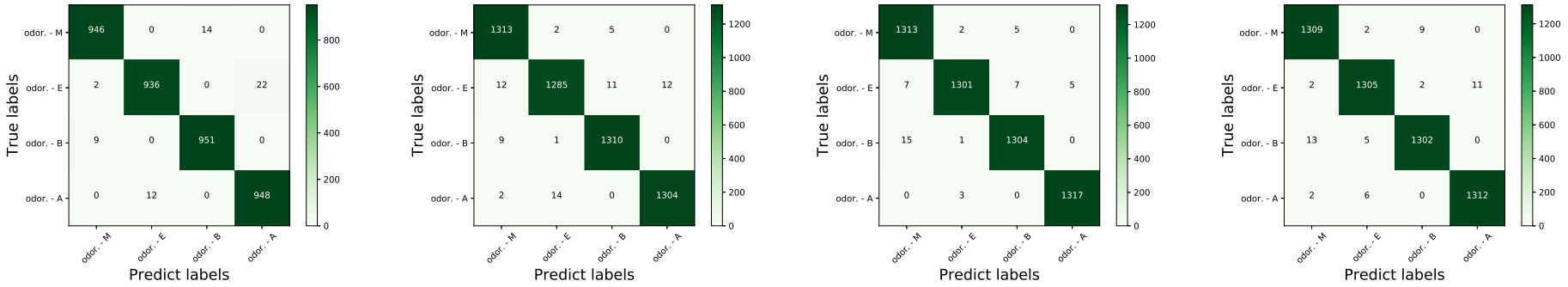


Figura D.25: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

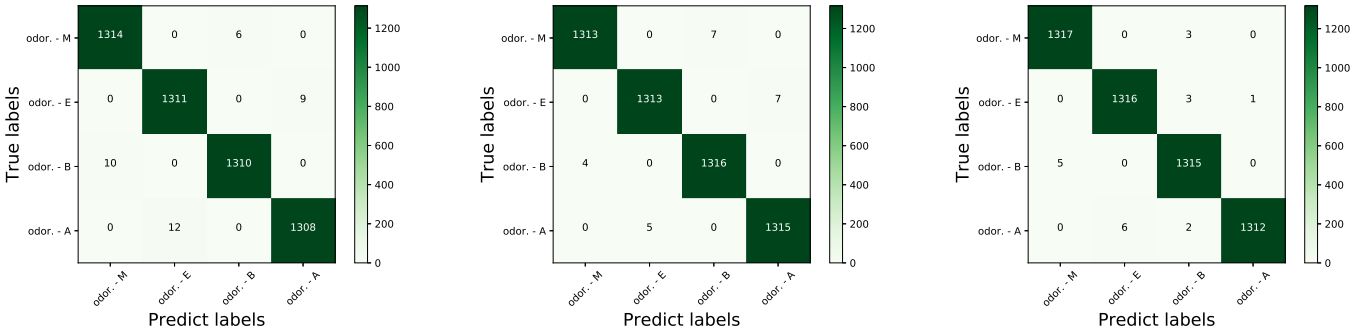


Figura D.26: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia y DS para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

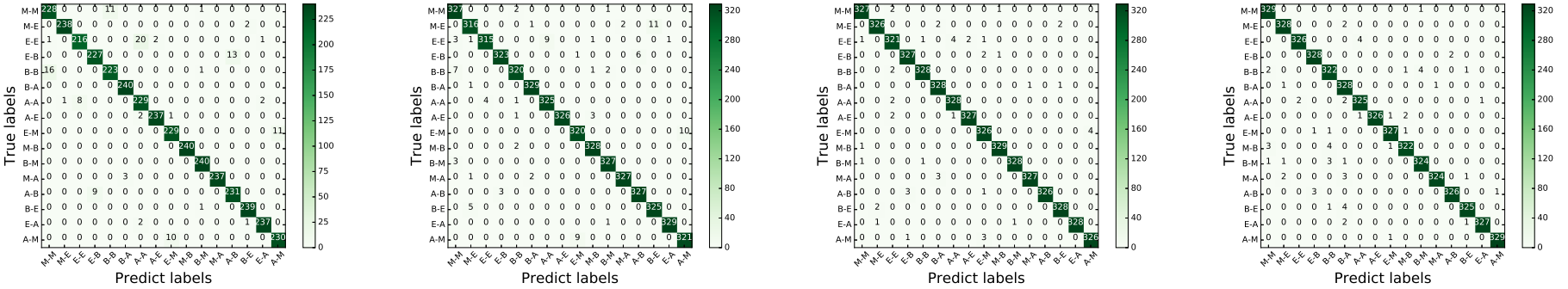


Figura D.27: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

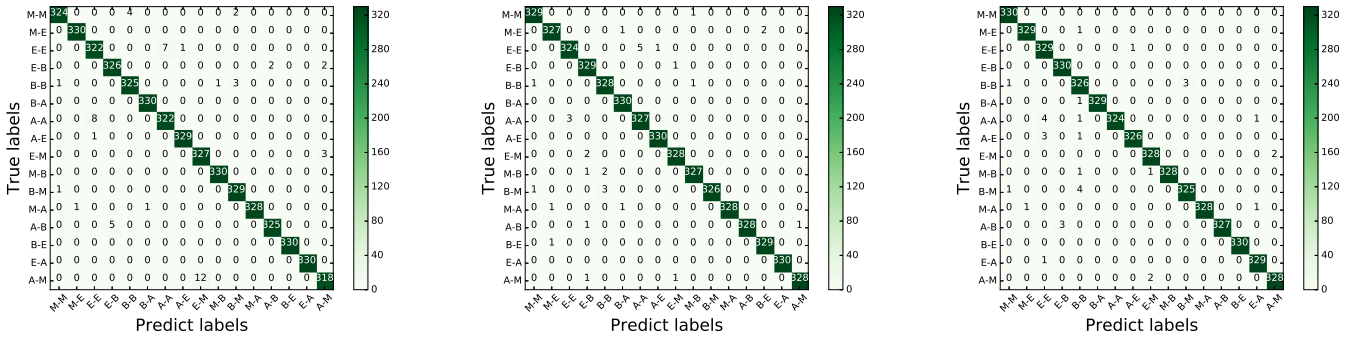


Figura D.28: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia y DS para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

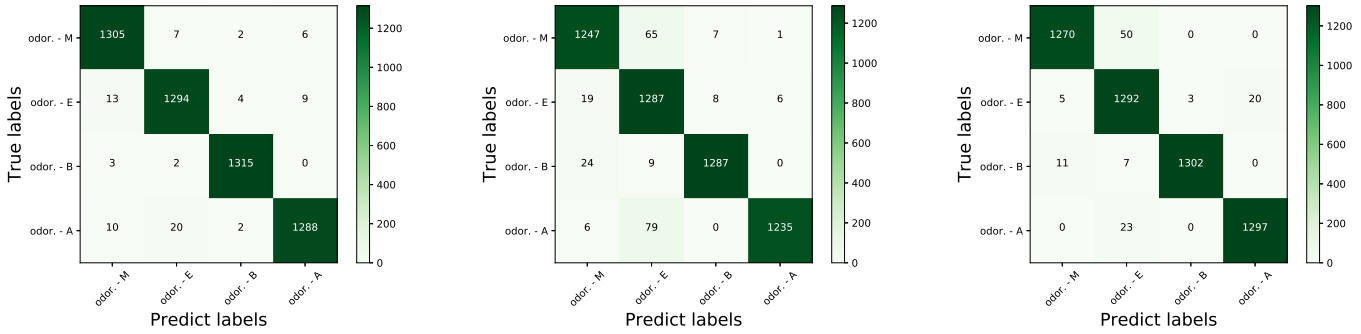


Figura D.29: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la temperatura para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

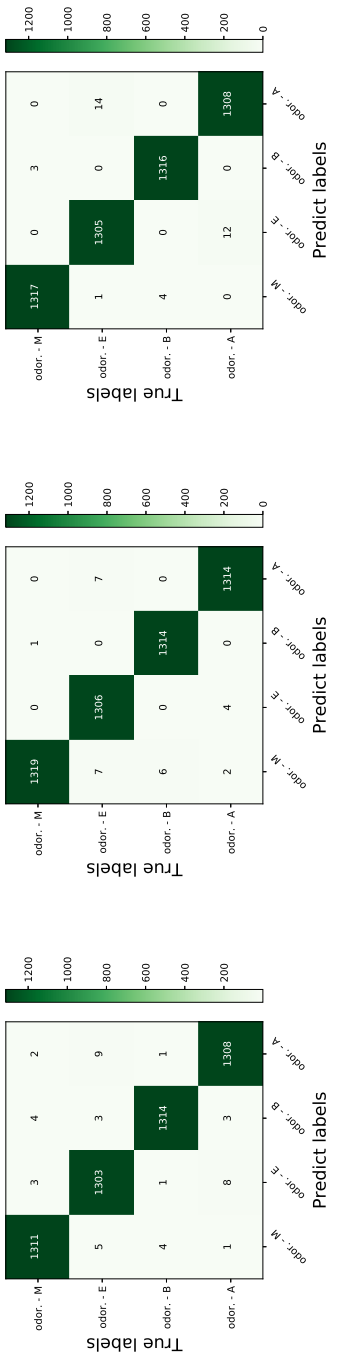


Figura D.30: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la temperatura y DS para el enfoque de los odorantes. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

Figura D.31: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la temperatura para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

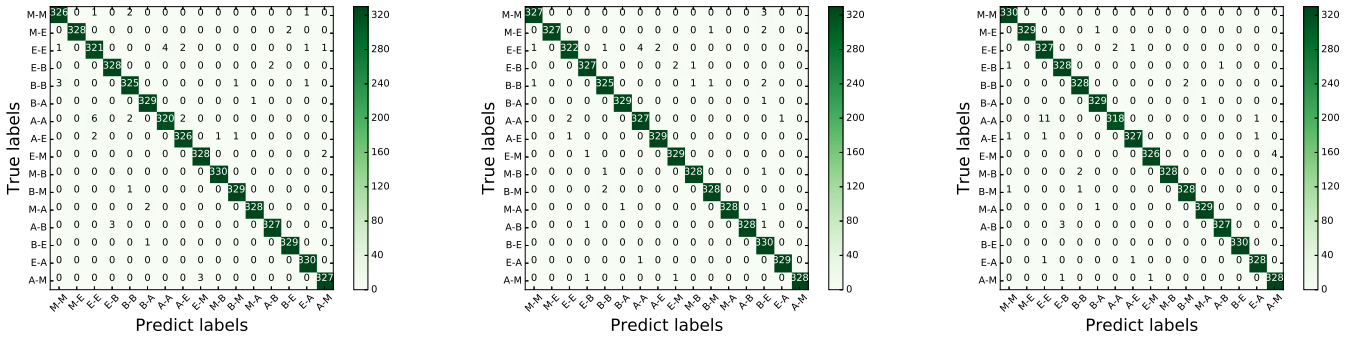


Figura D.32: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la temperatura y DS para el enfoque de las transiciones. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

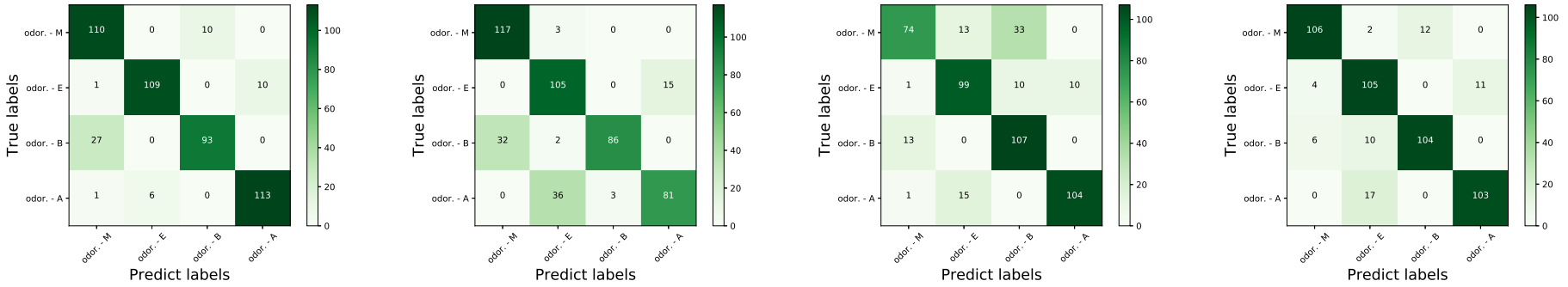


Figura D.33: Matriz de confusión obtenida de la clasificación mediante el uso de EMA para el enfoque de los odorantes aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

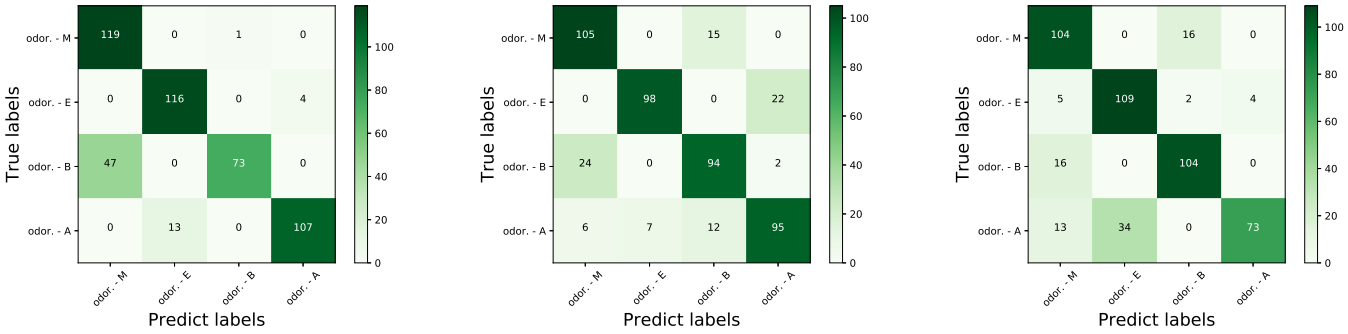


Figura D.34: Matriz de confusión obtenida de la clasificación mediante el uso de EMA y DS para el enfoque de los odorantes aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

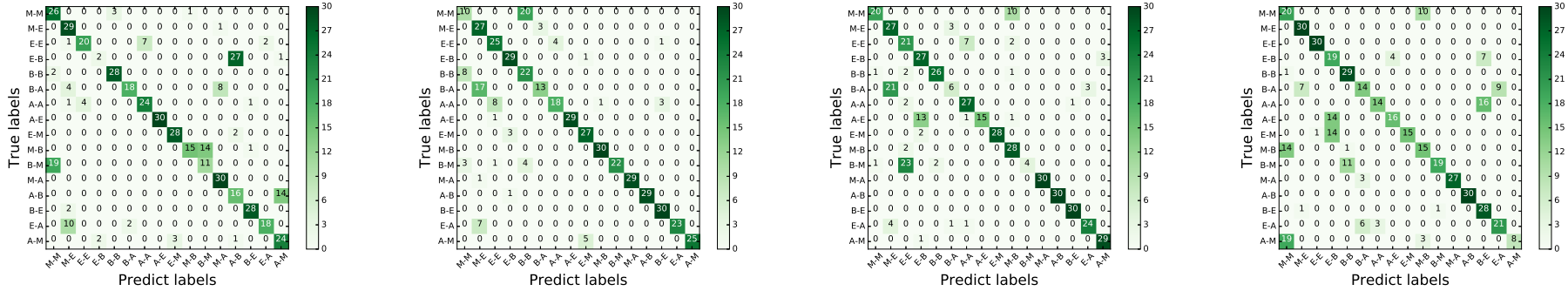


Figura D.35: Matriz de confusión obtenida de la clasificación mediante el uso de EMA para el enfoque de las transiciones aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

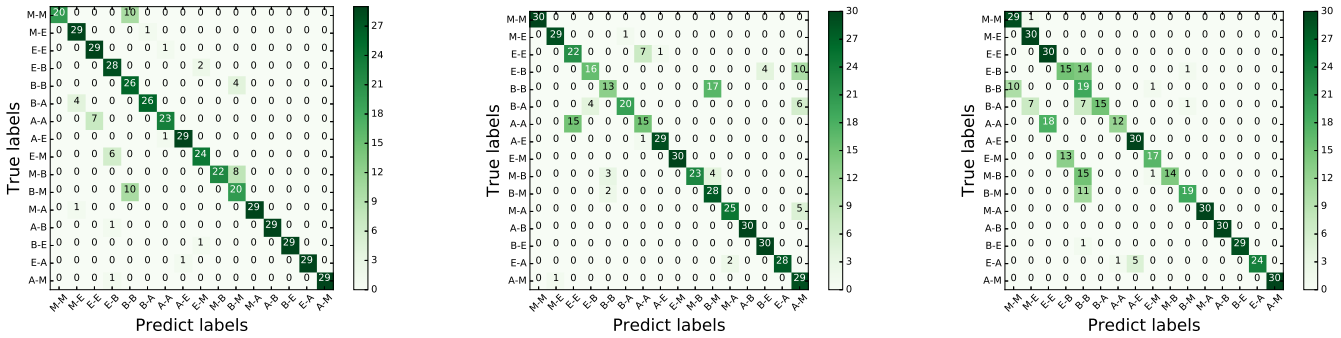


Figura D.36: Matriz de confusión obtenida de la clasificación mediante el uso de EMA y DS para el enfoque de las transiciones aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 1, 2 y 3.

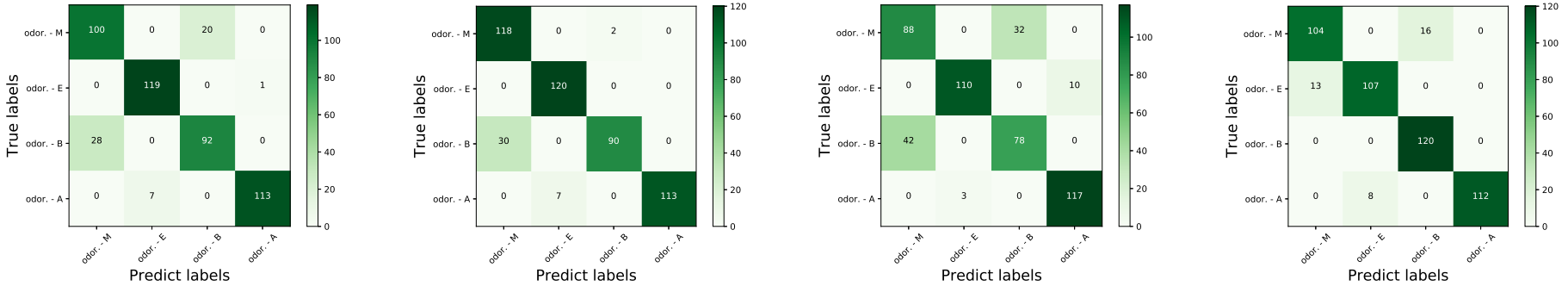


Figura D.37: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia para el enfoque de los odorantes aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.

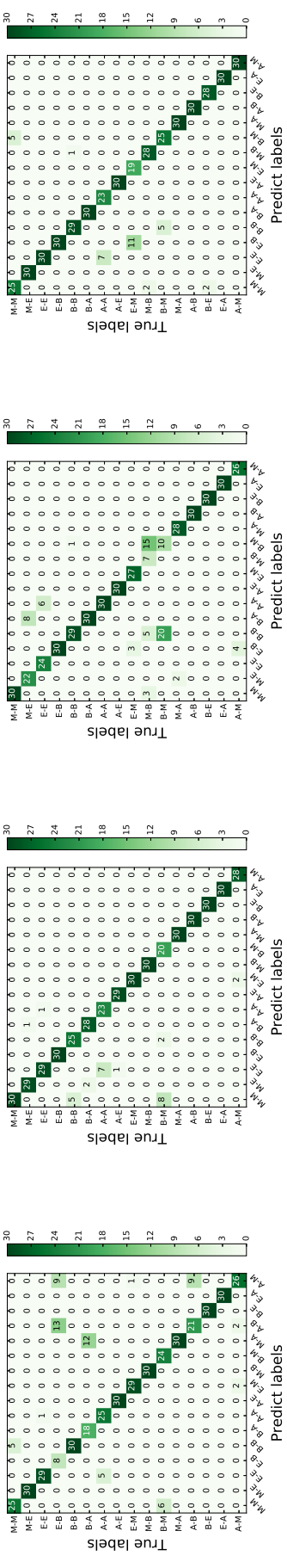


Figura D.38: Matriz de confusión obtenida de la clasificación mediante el uso de la huella de la resistencia para el enfoque de las transiciones aplicando el paradigma temporal. De izquierda a derecha se muestran las tendencias: 0, 1, 2 y 3.